ISSN 0265-2919

90p

64

THE HOME COMPUTER ADVANCED COURSE

MAKING THE MOST OF YOUR MICRO

I AM THE BROTHER HR-5.
MY GREATEST ADVANTAGE IS THE ABILITY TO PRODUCE UP 132 COLUMNS OF TEXT ONTO A SI
NGLE LINE.
THIS ALLOWS 'SERIOUS' APPLICATIONS SUCH AS WORD PROCESSING TO BE PERFORMED.
brother
HR-5
ALARM
PAPER EMPTY
RIBBON END
LINE FEED
ON LINE

An ORBIS Publication

IR£1.15 Aus $2.15 NZ $2.65 SA R2.45 Sing $4.50

CONTENTS

APPLICATION

STRATEGIC PLANNING The development of programs for games that require strategic play — such as chess and checkers — has always been at the forefront of AI research 1268

HARDWARE

HOT METAL Buying a printer for the Sinclair Spectrum can be a tricky business. We give some handy hints 1261

SOFTWARE

DYNAMIC CONTROL The Control Program for Microprocessors, one of the earliest and most highly regarded operating systems, is being increasingly implemented on home micros. We begin a new series that takes a detailed look at the program 1264

COMPUTER SCIENCE

THE PROLOGUE Our introduction to a short tutorial course on PROLOG explores some of its fundamental logical concepts 1272

JARGON

FROM PASSWORD TO PATTERN RECOGNITION A weekly glossary of computing terms 1280

PROGRAMMING PROJECTS

FAIR EXCHANGE The purpose of our trading expedition is upon us: we must now barter our goods (if we have any left) with the inhabitants of the New World 1266

MACHINE CODE

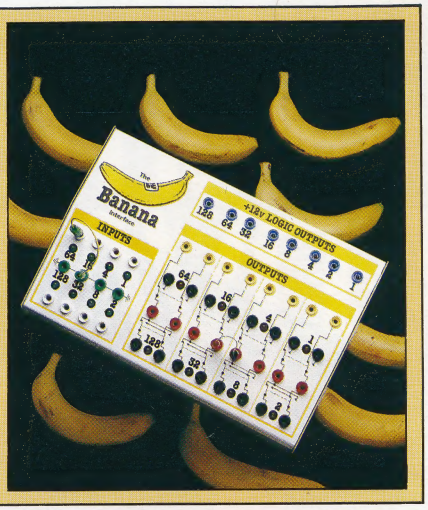
GRAPHICS REVOLUTION We conclude our exploration of ways to use the Commodore 64 operating system calls in our machine code programs. The 3-D graphics package we began last week is completed 1276

WORKSHOP

SETTING YOUR SIGHTS The construction of the digital tracer for the BBC Micro is completed. Software to test and control the device will be given next week 1274

Next Week

- **Our investigation of the CP/M operating system starts to explore some of the commands that are used to manage disk files.**
- **To begin a major new series on the Sinclair Spectrum's OS, we take a look over the computer's memory map.**
- **Our New World simulation game sets sail on the home stretch, and the final reckoning of profit or loss.**



QUIZ

1) What are the minimum hardware requirements to allow a computer to run CP/M?
2) What peripheral is required to drive a Brother HR-5 printer from the Spectrum?
3) What problems are encountered in the 'template' method of pattern recognition?

Answers To Last Week's Quiz

1) By simply using a 'hill climbing' strategy to find the centre of a maze, we are likely to find ourselves in a dead end which is not the centre. We could not retrace our steps to find another solution, because this would take us away from the centre.
2) When the tracer's arm is moved a voltage in the potentiometer is produced, which can be decoded by the computer to calculate the arm's new position.
3) Yes. What are the same, however, are the input/output call addresses which have to be identical for complete compatibility.
4) In order to retrieve a BASIC string, the Commodore stores the start address of the string and its length in bytes.

Editor Stephen Cooke; **Art Editor** Claudia Zeff; **Production Editor** Bobby Pickering; **Deputy Editor** Steve Colwill; **Designer** Julian Dorr; **Art Assistant** Liz Dixon; **Staff Writer** Steve Malone; **Sub Editor** Jon Kaye; **Contributors** Richard Forsyth, Graham Storrs, Dr Antonia Jones, Terry Barrett, Steve Malone, Anthony Ginn, Steve Colwill; **Software Consultants** Pilot Software City; **Group Art Director** Perry Neville; **Managing Director** Stephen England; **Published by Orbis Publishing Ltd: Editorial Director** Brian Innes; **Project Development** Peter Brookesmith; **Executive Editor** Maurice Geller; **Production Assistant** Alastair Gourlay; **Subscription Manager** Christine Allen; **Designed and produced by Bunch Partworks Ltd; Editorial Office** 14 Rathbone Place, London W1P 1DE; **© APSIF Copenhagen 1985; © Orbis Publishing Ltd 1985: Typeset by Universe; Reproduction by Mullis Morgan Ltd; Printed in Great Britain by Heanor Gate Printing Ltd, Derby**

HOW TO OBTAIN ISSUES AND BINDERS FOR THE HOME COMPUTER ADVANCED COURSE – Issues can be obtained by placing an order with your newsagent or direct from our subscription department. If you have any difficulty obtaining any back issues from your newsagent, please write to us stating the issue(s) required and enclosing a cheque for the cover price of the issue(s). **AUSTRALIA** – please write to: Gordon & Gotch (Aus) Ltd, 114 William Street, PO Box 767G, Melbourne, Victoria 3001. **MALTA, NEW ZEALAND & SOUTH AFRICA –** Back numbers are available at cover price from your newsagent. In case of difficulty, write to the address given for binders. **UK/EIRE** – Price: 90p/IR£1.15. Subscription: 6 months: £26.00. 1 Year: £52.00. Binder: please send £3.95 per binder, or take advantage of our special offer in early issues. **EUROPE** – Price: 90p. Subscription: 6 months air: £44.72. Surface: £36.14. 1 year air: £89.44. Surface: £72.28. Binder: £5.00. Airmail: £8.25. **MALTA** – Obtain binders from your newsagent or Miller (Malta) Ltd, MA Vassalli Street, Valetta, Malta. Price: £3.95. **MIDDLE EAST** – Price: 90p. Subscription: 6 months air: £50.18. Surface: £36.14. 1 year air: £100.36. Surface: £72.28. Binder: £5.00. Airmail: £8.25. **AMERICAS/ASIA/AFRICA** – Price: US/CAN$1.95/90p. Subscription: 6 months air: £59.54. Surface: £36.14. 1 year air: £119.08. Surface: £72.28. Binder: £5.00. Airmail: £9.50. **SOUTH AFRICA** – Price: SA R2.45. Obtain binders from any branch of Central News Agency or Intermag, PO Box 57394, Springfield 2137. **SINGAPORE** – Price: Sing $4.50. Obtain binders from MPH Distributors, 601 Sims Drive, 03-07-21, Singapore 1438. **AUSTRALASIA/FAR EAST** – Price: 90p. Subscription: 6 months air: £64.22. Surface: £36.14. 1 year air £128.44. Surface: £72.28. Binder: £5.00. Airmail: £9.75. **AUSTRALIA** – Price: Aus$2.15. Obtain binders from First Post Pty Ltd, 23 Chandos Street, St Leonards, NSW 2065. **NEW ZEALAND** – Price: NZ$2.65. Obtain binders from your newsagent or Gordon & Gotch (NZ) Ltd, PO Box 1595, Wellington.
ADDRESS FOR BINDERS AND BACK ISSUES – Orbis Publishing Limited, Orbis House, Bedfordbury, London WC2 4BT. Telephone 01-379 5211. Cheques/postal orders should be made payable to Orbis Publishing Limited. Binder prices include postage and packing and prices are in stirling. Back issues are sold at the cover price, and we do not charge carriage in the UK.
NOTE – Binders and back issues are obtainable subject to availability of stocks. Whilst every attempt is made to keep the price of the issues and binders constant, the publishers reserve the right to increase the stated prices at any time when circumstances dictate. Binders depicted in this publication are those produced for the UK and Australian markets only. Binders and Issues may be subject to import duty and/or local taxes, which are not included in the above prices unless stated.
ADDRESS FOR SUBSCRIPTIONS – Orbis Publishing Limited, Hurst Farm, Baydon Road, Lambourn Woodlands, Newbury Berks, RG16 7TW. Telephone: 0488-72666. All cheques/postal orders should be made payable to Orbis Publishing Limited, Postage and packaging is included in subscription rates, and prices are given in sterling.

COVER PHOTOGRAPHY BY CHRIS STEVENS

CHRIS STEVENS

The Four Degrees
Over the past few years a number of thermal printers have come onto the market. Although some machines, such as the Floyd 40 and the Alphacom 32, are dedicated especially for use with the Sinclair Spectrum, other machines, such as the Epson P40 and the Brother HR-5 are intended for use with a wide range of home micros

HOT METAL

Because the Sinclair Spectrum's interfaces tend not to accommodate most printers, the only option for Spectrum owners has been the ZX Printer, not known for its endurance or print quality. We present here four alternatives for the Spectrum, as well as listing several major criteria to be considered when shopping for a printer.

One of the first peripherals that computer owners consider buying is a printer, which will enable them to produce 'hard copy' listings of their programs. But for Sinclair Spectrum owners, there are a number of problems in choosing a printer. On the one hand, the Spectrum lacks any of the conventional interfaces that are generally used by printers, and on the other, Sinclair Research's own ZX Printer, now discontinued, was noted for its messy print and a tendency for the copy to fade with time. Because Sinclair Research has chosen not to upgrade its printer, other manufacturers have produced budget-priced thermal printers of their own. Here, we examine some of these alternatives.

Several basic considerations have to be borne in mind when shopping around for a printer. Primarily, of course, there is the cost — a consideration that often involves a lot more than just comparing price tags. There may be hidden costs, for example, that are not included in the price of the machine. A printer's manufacturer may state that it is designed for a Spectrum, when in fact it requires an RS232C interface socket. Therefore, before the printer could be used, an Interface 1 extension (which connects an RS232C socket to the Spectrum) would also have to be purchased.

HIDDEN COSTS

Another hidden cost is printer paper. Many printers will only use paper specifically designed for them. For example, thermal-type printers like the ZX Printer itself require special heat-sensitive paper. Therefore, after paying for your printer, you will effectively be tied to the manufacturer's whims, and the paper may be difficult to obtain and overpriced.

When buying a printer, it is therefore essential to check whether the shop from which the device is sold keeps a regular supply of paper and other service items, and to know how much they are likely to cost. You should also enquire about servicing. Machinery having mechanical moving parts, such as the print head, is far more prone to failure than the electronic components and thus it is preferable to purchase your printer from a company with an efficient servicing department.

Finally, there is the problem of software compatibility to consider. All printers are programmed to respond to codes (usually ASCII

characters) which tell them to perform certain functions like carriage return, number of characters per line and print size. Although many of these will be identical to those used by the Spectrum, this may not be true of every machine.

Sinclair, like most other home micro manufacturers, has customised many of the ASCII codes for its own uses and, although the printer manual might indicate a code meaning one thing, it may mean quite another to the Spectrum. The solution here is to see the printer's facilities running from a Spectrum before you buy.

The same precautions apply to any software that you might wish to use. There is nothing quite so frustrating as command codes that operate quite happily from BASIC but which cannot be used from within a particular word processing package. Thus you may find yourself in the unenviable position of having to SAVE your copy to tape or disk, switching off and reading the copy back as a sequential file in order to print it out.

Having set several criteria for purchasing a printer for the Spectrum, let's now look at some of the low-cost thermal printers that are available. Among the printers shown, the cheapest is the Alphacom 32. It is also the machine that most closely resembles the ZX Printer. The founts used are like those of the ZX Printer and, as with the Sinclair machine, it can only print a maximum of 32 columns.

For some reason, the manufacturers have chosen to equip the Alphacom with its own power supply. The addition of this extra box and accompanying wires is unnecessary, especially as the Spectrum's expansion bus has a 9v supply, and this should be sufficient to run a thermal printer.

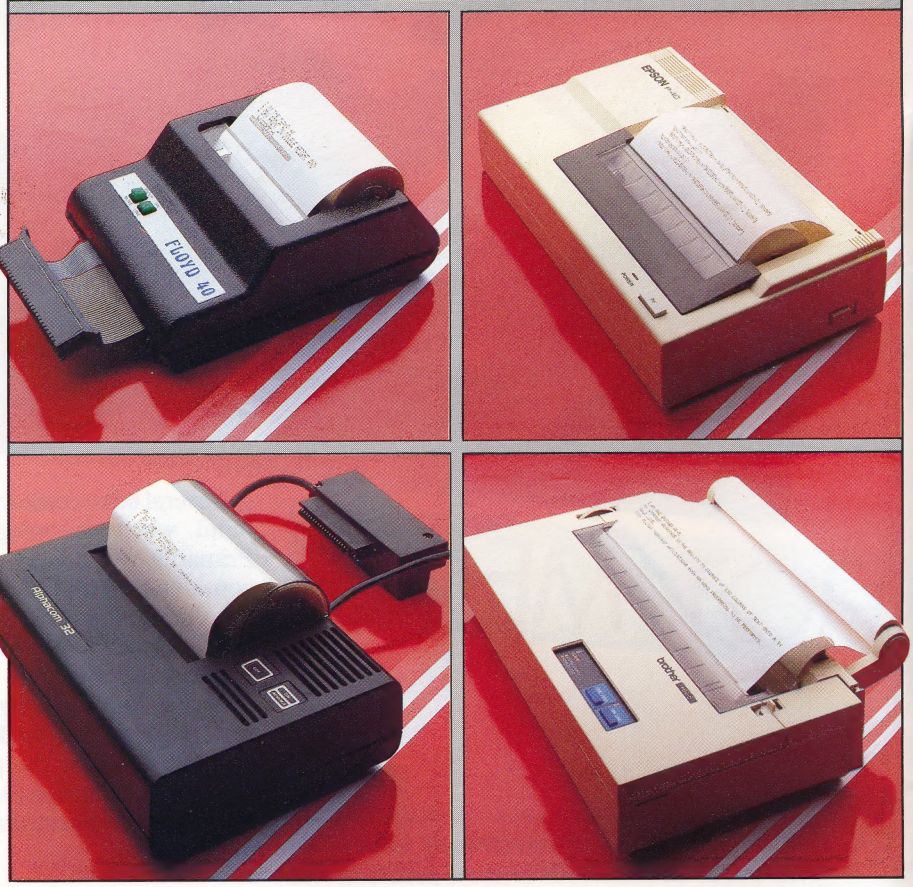
The construction of the Floyd 40, from Shiva Marketing, appears to be somewhat flimsy. The casing bends fairly easily and the roller on which the reel of paper is held is merely a thin wooden dowel. That said, however, the machine is a marked improvement over the Alphacom.

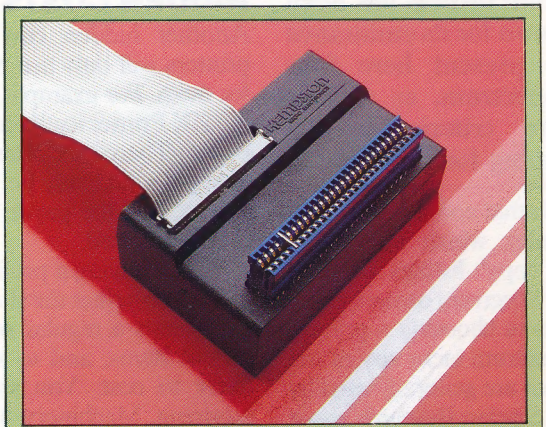
Power is drawn from the expansion bus of the Spectrum and so there are no extra boxes needing to be attached. The Floyd 40 also uses white paper, which makes the black print much more legible. What really sets this machine apart from the Alphacom and the ZX Printer, however, is that it accepts a number of control characters that can alter the output to the printer.

These characters are sent to the printer via a normal LPRINT command followed by quotes. The control character itself is enclosed in exclamation marks, informing the printer not to print it because it is a command. For example, the line LPRINT "!H!" will order the Floyd 40 to print in double-height characters. Issuing the command a second time cancels the formatting.

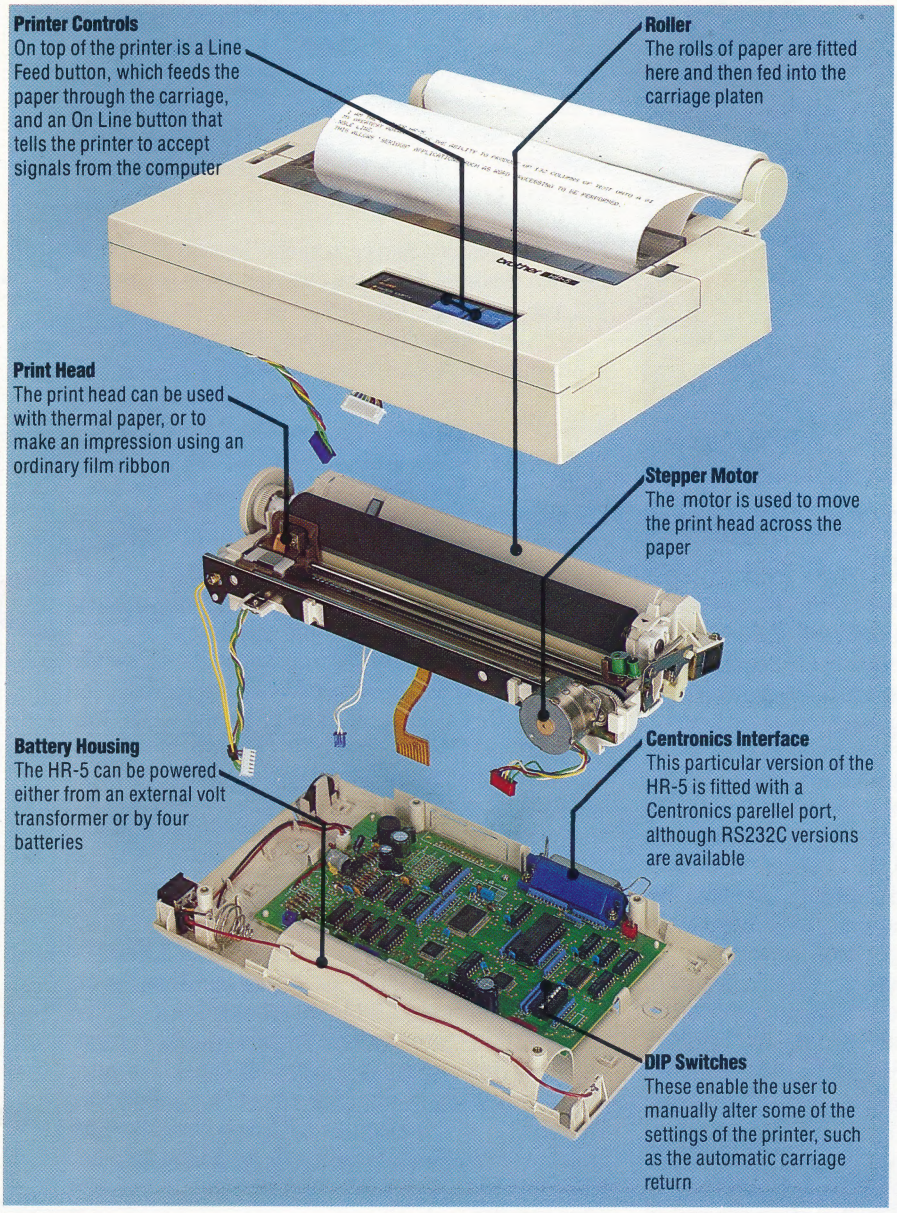
Other facilities that are available include graphics characters, wordwrap, double-width and inverted format. The five by seven-pin print head produces attractive pictures in graphics mode and the print format is as good as many other machines costing twice as much. Although fairly slow, the

MODEL	Alphacom 32	Floyd 40	Epson P40	Brother HR-5
PRICE	£54.95	£69.95	£99.95	£149.95
PAPER COST	£10.95 (3 rolls)	£2.50 (per roll; £12.50 for 10 rolls)	£8.65 (5 rolls)	£4.75 (per roll)
INTERFACE USED	Expansion bus	Expansion bus	RS232 or Centronics parallel	RS232 or Centronics parallel





Kempston Interface
In order to use some of the more sophisticated printers (which are equipped with Centronics or RS232C interfaces), it will be necessary for the Spectrum owner to invest in a suitable peripheral device that provides these connections. Without doubt, the most popular such interface on the market is the Kempston Interface, which provides access to a wide range of Centronics-compatible printers


Printer Controls
On top of the printer is a Line Feed button, which feeds the paper through the carriage, and an On Line button that tells the printer to accept signals from the computer
Roller
The rolls of paper are fitted here and then fed into the carriage platen
Print Head
The print head can be used with thermal paper, or to make an impression using an ordinary film ribbon
Stepper Motor
The motor is used to move the print head across the paper
Battery Housing
The HR-5 can be powered either from an external volt transformer or by four batteries
Centronics Interface
This particular version of the HR-5 is fitted with a Centronics parellel port, although RS232C versions are available
DIP Switches
These enable the user to manually alter some of the settings of the printer, such as the automatic carriage return

CHRIS STEVENS

only real drawback to the system is that it is restricted to an 80mm (3⅛in) width roll of paper.

The third machine shown is the Epson P40, which is sold in two basic versions covering the main printer interface types. The P40S is a serial version with an RS232C socket, whereas the P40P has a Centronics parallel port on the back. However, as the Spectrum has neither of these fitted as standard, you must buy either an Interface 1 or one of the many other interfaces on the market.

There is a problem in that the RS232 socket supplied by Sinclair on the Interface 1 is a non-standard seven-socket D connector. The manuals for the P40 and the Interface 1 will show you which way the pins should be positioned and from there it is a simple soldering job. If, however, you are averse to using soldering irons, it's perhaps better to buy one of the third party interfaces that are available. Kempston, for example, provides interfaces that plug into the back of the Spectrum as well as standard connections for both RS232 and Centronics devices.

The Epson P40, despite its relatively small size, can print in either 40 or 80 column mode. The machine can also make use of a wide number of escape codes that are available on the larger devices in the range. Of course, the Spectrum does not have an Escape key *per se* so that the codes would have to be sent as ASCII codes in the format CHR$(27);"E"; (where CHR$(27) represents the ASCII Escape character) which tells the P40 to set emphasised mode.

The P40 can also perform a number of other operations, such as altering the character set, entering bit image mode (which allows you to create your own characters) and condensed mode. Apart from these alterations, which can be performed from software, there are also DIP switches, enabling you to set the parity and standard number of print columns.

Because the P40 carries the prestigious Epson brand name, you can feel quite assured that there will be sufficient support around for some time to come. The only major drawback is that it costs almost £100, so that once you have bought an interface (for another £60 or so) the cost of the printer system begins to exceed the cost of the computer itself.

Unlike the other printers featured here, the Brother HR-5 can use either thermal paper or, if it is fitted with a ribbon, normal paper. Another advantage of this machine over the others is that the carriage can accept A4-sized paper and so can be used for letters and other standard word processing applications. The additional width allows the Brother to print up to 132 characters per line.

Like the Epson machine, the HR-5 makes extensive use of Escape codes to format its printing, and is thus able to produce a wide range of characters, international founts and other formats. It is also nearly silent in operation. Another similarity the Brother HR-5 shares with the Epson is that, unfortunately, the machine is fitted with either a Centronics parallel interface or an RS232C socket. Once again, you would therefore have to buy a suitable interface in order to run the printer from a Spectrum. Apart from this, the quality of the output is excellent.

In choosing a printer, problems occur only if you decide to buy a machine whose capabilities extend beyond those of the machines custom-built for the Spectrum. The Alphacom and the Floyd 40 have the advantage of simply plugging into the computer and will accept normal Spectrum commands like COPY, but their capabilities are really just limited to listings.

Printers higher up the price range, with a wider choice of applications, are designed to work with a variety of different machines, will not be customised to work with the Spectrum, and so will not be as friendly. Until someone comes up with a high quality printer designed specifically for the Spectrum, you will, unfortunately, have to adapt the printer to the computer.

DYNAMIC CONTROL

SOL System
£2,500

We begin a series of articles in which we will take a detailed look at the development, structure and operation of one of the most popular operating systems — CP/M. Having built up a formidable reputation with business users over the past decade, this operating system is being increasingly implemented on home computers.

When floppy disk storage systems were invented in the early 1970s, there was obviously a pressing need for the development of an operating system that would be able to organise the information held on these devices without users having to do so 'manually'. At about this time, the American microchip manufacturer Intel decided to develop a floppy disk system for its recently introduced 8000 series of microprocessors. The company's development team was headed by a young consultant engineer by the name of Gary Kildall. The resulting program became known as CP/M (originally an acronym for Control Program/Monitor, later changed to Control Program for Microprocessors).

When the package was launched in 1975 it quickly became a runaway success. In fact, it became so successful that when a group of designers left Intel to form their own company, Zilog Inc, they decided to make their new chip — the Z80 — compatible with the 8008 so that it could run CP/M. Kildall himself also left Intel to form his own company, Digital Research, which is now one of the largest software companies in the world.

The success of CP/M can be measured by the fact that although many rival operating systems appeared for eight-bit micros during the 1970s, particularly those based around the non-CP/M-compatible 6502 processor, the system is now the de facto industry standard for eight-bit business micros. This is especially astonishing considering that CP/M is over a decade old — in microcomputing terms, an extraordinary lifespan.

To be able to run CP/M, you need a computer system with an 8008 microprocessor or a compatible chip, a floppy disk drive and at least 48 Kbytes of memory available. This large quantity of RAM is needed because often the memory will be required to hold the CP/M program itself and any of the command files (which can use up to eight Kbytes on their own) and disk files on which it may be operating. With the advent of 16-bit computers, it seems the heyday of CP/M has now passed in the business market.

But the system has been adopted by many home and educational computer manufacturers, allowing their Z80 micros to break out of the games market and move into more 'serious' applications. The addition of a CP/M-compatible disk drive instantly provides the consumer with a huge quantity of software that was developed for 8008 and Z80-based business micros in the 1970s. This lets the manufacturers sidestep the problem of software support that has plagued many micros. Manufacturers that have taken this route in the past few years include Memotech, Research Machines and Amstrad.

Unlike most home computers, which have their operating systems installed 'on board' (and which are activated on power-up), CP/M is nearly always resident on a floppy disk (known as a 'systems disk') which is loaded from a disk drive on power-up. There is no particular reason why CP/M should be held on a systems disk as some companies have provided similar systems on ROM. The reason is mostly historical. On early computers, memory was at a premium; if CP/M was installed but not needed, it would only have taken up precious memory which could have been used for other purposes. Thus, it was kept on disk and LOADed only when required.

Typically, a CP/M system disk contains the CP/M program itself and a number of smaller programs that can be called via CP/M to perform specific disk-handling functions. CP/M is more than simply a disk filing system, however; it is a complete operating system in itself.

When CP/M is loaded into a computer with its own operating system resident in ROM, it will 'page out' the machine's system and take over the complete running of the computer. Thus, when the computer is running under CP/M, BASIC commands that would normally be executed by the computer will not be recognised by CP/M and will generate a syntax error. In fact, a BASIC program will not run at all under CP/M unless the operating system has a BASIC interpreter or compiler added to the system.

On power-up, the CP/M systems disk will generally be run automatically and a directory of some of the commands available will be displayed (as shown). A

close examination of the disk directory tells us a number of things about the way CP/M is structured. The series of commands (like all files in CP/M) consists of two parts, beginning with the 'primary' name. When we wish to LOAD any command file, we simply type in this part of the file's full name together with the name of the file on which we want to operate, and it will be LOADed and RUN automatically.

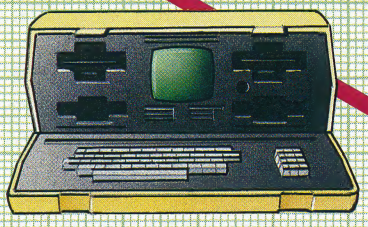
The second part of the full filename (to the right of the full stop) is known as the extension, which tells CP/M (and you) what kind of file it is. Because all the files on a CP/M systems disk are command files, they are all suffixed with the extension COM. We shall be looking at other kinds of file extension later in the series.

Another thing noticeable when examining the disk directory is that the CP/M program itself is not listed. At first glance, therefore, it may appear that CP/M is merely the sum of its parts and a collection of smaller command programs. Although this is the way that CP/M is structured to a large extent, a closer look at the directory shows that while many vital commands (such as LOAD) are present, others (such as SAVE) are missing. Clearly the directory is not telling us the whole story.

```
MRENMENU version 1.1
(c) 1984 Quantec Systems and Software Ltd
A)dir
A: MRENMENU COM : CPM3     SYS : COPYSYS  COM : ERASE    COM : SET      COM
A: SHOW     COM : SUBMIT   COM : TYPE     COM : DIR      COM : GET      COM
A: PUT      COM : RENAME   COM : SETDEF   COM : PIP      COM : HELP     COM
A: HELP     HLP : FC       COM : DEVICE   COM : FORMAT   SUB : FC       GET
A: FORMAT   GET : CONFIGUR COM
A>
```

Looking Up The Directory
Typing DIR will display a list of the files that are held on the CP/M system. Unless you have already added some files to the disk, all the entries will be COMmand files, identifiable by the .COM suffix on the end of each file name. Other information is also displayed by the directory, such as how much memory is taken up by each file and the name of the file from which the directory was accessed

Osborne 1
£1,695



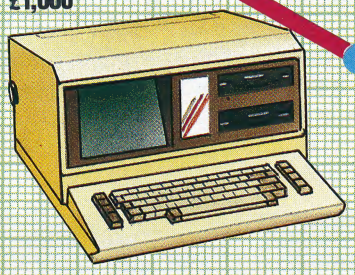
We noted previously that an operating system should, as far as possible, be transparent, creating as little work as possible for you in order to manage the system properly. When we boot the systems disk, CP/M is loaded into the user RAM and remains there while the computer is switched on. A number of commonly used command files (known as utility programs) which also reside in RAM are LOADed with the main CP/M program as well. Thus, to use these commands, there is no further need to LOAD them from the disk.

To prevent any confusion that may arise in your mind as to which commands are already resident in RAM and which have to be LOADed, these commands, like the CP/M program itself, have been made 'invisible' to the directory and are known as 'built-in' commands. Command files held on disk are known as 'transient' commands. Once executed they are discarded by the operating system and have to be reLOADed if you wish to use them again.

The reason for this apparently arbitrary system of executing commands is partly due to hardware restrictions and partly because it adds flexibility to the system. As CP/M is based around eight-bit processors, the maximum memory that can be addressed directly by the processor is 64 Kbytes. If all of the commands necessary to run CP/M were loaded into memory on power-up, there would be very little user RAM left for programs and text files. Thus it was decided that while as many commands as possible should be held in RAM, memory would have to be kept on disk and discarded when used, in order to conserve memory.

However, such a system does have its advantages. It means that extra transient commands can be added quite simply by adding them to the directory with a .COM extension. This flexibility in being able to add extra commands that can be run under the general umbrella of CP/M has resulted in the enormous quantity of applications programs that have been written to be run under CP/M. As an example, the popular word processing package, WordStar (although it is treated as a separate program that can be run under CP/M), is regarded by CP/M as another of its command files when it is loaded into the system.

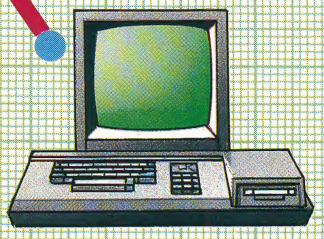
Wren
£1,000



This in itself produces several advantages. To begin with, WordStar can be used on any CP/M system — the program does not require its own disk-handling routines as it can simply call up CP/M's commands to perform these functions. This means that WordStar is a considerably smaller program than it would otherwise have to be and takes up far less memory that can consequently be used to hold text files.

Having looked at the development and structure of CP/M, the next instalment will deal with some of the utility programs that are available and how they work.

Amstrad CPC 664
£339



FAIR EXCHANGE

Providing the game has not ended through misfortune or mismanagement, you will have arrived safely in the New World and be able to begin trading. With a bit of keen trading sense, you might acquire carvings, spice and pearls that could bring you a healthy profit when you return home.

The trading routine is called from the main program section at line 892 and its first task is to check whether you have brought any guns with you to trade. Although guns can be useful — to ward off pirates and so on — the chief of the New World inhabitants has developed a dislike for the weapons, knowing full well they could cause problems among his people. This means, unfortunately, that if you *have* brought guns for bartering with, the chief will decline the offer and a message will be displayed to that effect. The remaining goods, however, are traded in lots.

The routine continues, and informs you what price each of the goods offered by the chief commanded at home before you set sail. Salt is traded first and you may trade it for pearls, carvings or spice. After the salt has been traded, you can set up a deal for the cloth in the same manner, followed by the knives and the jewels.

It will not be necessary for the chief to tell the captain that the guns are not wanted if there are none on the ship — so line 10072 looks at the second element of the supplies array, OA(2), in which the number of guns is recorded, and ignores the chief's message if this element is set to zero.

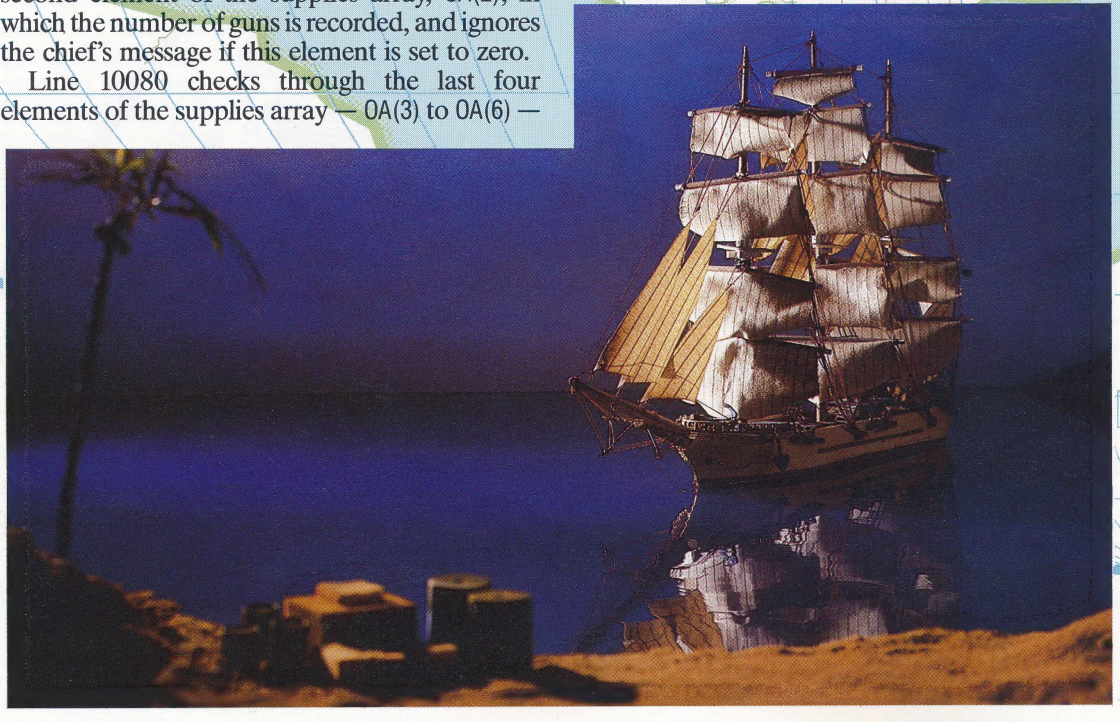
Line 10080 checks through the last four elements of the supplies array — OA(3) to OA(6) — for goods suitable for trading. The first two elements represent medicine and guns, which are unsuitable and are therefore not checked. Should there be any knives, salt, cloth or jewels remaining, you will be told that the chief is willing to offer pearls, carvings and spice in exchange. But should all four elements in the array be set to zero, the game will end because, obviously, there will be no goods to trade.

After informing you of the value of each of the goods when the ship left port, and warning you that the price may have changed since then, the program enters a loop between lines 10130 and 10200 to handle the mechanics of the trading. The code within this loop calculates the amounts of pearls, carvings and spice offered by the chief for each lot of merchandise you wish to trade and asks you which goods you want. It then stores the selected goods on the ship.

The loop counts from 3 to 6, thus ignoring guns and medicine (which are held in the first and second elements of OA()) and deals with each commodity separately. Line 10135 checks to see whether the value in the array element currently under examination is 0, signifying that none of that goods type is held by the player and, if so, moves on to the next element in OA() by jumping to the NEXT instruction at line 10200. Otherwise, you will be informed of the quantity of each, printed by line 10145. Lines 10150 to 10153 print the goods type, corresponding to the current value of T.

After the chief makes an offer of some pearls, carvings and spice, the program calculates the quantities of these commodities that you will receive for each of the goods you offer. The two-dimensional array EQ(,), created at line 63 and containing the exchange rates, is used for this computation.

The expression in line 10165 calculates the



amount of pearls offered for each item in the loop, multiplying the quantity of your exchangeable goods, OA(T), by the exchange rate set in array EQ(,). Because the loop is set at 3 to 6, 2 must be subtracted from T to synchronise it with the element numbers of the provisions in EQ(,), which are from 1 to 4. If the loop is dealing with salt, the loop counter T will equal 3; EQ(T-2,1) therefore relates to the intersection between the first element of the first subscript, representing salt, and the first element in the second subscript, representing pearls. In EQ(,), this is set at .5; so, for salt, line 10165 would multiply the amount of salt by .5 to calculate the number of pearls offered in exchange.

Line 10166 performs a similar function on the amount of carvings offered by the chief in exchange. The rate of exchange for carvings is recorded in the second element of the second subscript in array EQ(,). Each successive value of the loop counter T therefore directs the program to the appropriate section of the second array column. Similarly, line 10167 determines the amount of spice offered by the chief for each of your goods, using the third column of the exchange rate array.

CLINCHING A DEAL

After being told what the chief has offered, you must decide which goods you want to trade. 1, 2 or 3 must be entered for the type of goods required, and line 10176 determines if the input was correct by ensuring that the numeric range of the input lies in the range of 1 to 3. The exchanged goods must then be taken to the ship — line 10180 stores the amount in the array AO(), DIMensioned at line 68.

The three elements of this array represent the three goods the chief has to offer. The amount of goods taken onto the ship is again calculated using the exchange rate array EQ(,) multiplying the appropriate element by the amount of goods traded, OA(T). I, the number you have to type to select the desired category of goods, is used to select the correct element of AO(), in which the traded pearls, carvings and spice are to be stored. I is also used as the second subscript of EQ(,).

Note that if a second amount of pearls, carvings or spice is obtained in subsequent trading, the amount must be added to that already stored in the array AO(). After the newly obtained merchandise has been recorded in AO(), the loop returns, if necessary, to the start. When all the goods have been exchanged, trading ceases and you will be told of the amounts of pearls, carvings and spice that have been acquired and placed on board your ship. Lines 10220 and 10244 print the amount of each commodity, which are now stored in array AO(). Control will then return to the main program.

In the next, and concluding, instalment, we complete the game by adding the final sections that will involve you with a local insurrection before sailing home and selling your newly acquired goods, hopefully to make sufficient profit to justify your voyage.

Module Twelve: Trading

Addition To Main Program Body

```
892 GOSUB10070
```

Trading Routine

```
10070 PRINTCHR$(147):GOSUB9200:REM TRADING
10072 IFOA(2)=0THEN10080
10074 S$="THE CHIEF DOESN'T WANT YOUR*":GOSUB9100
10076 S$="GUNS - THEY WILL CAUSE TROUBLE*":GOSUB91
00
10078 PRINT:GOSUB9200
10080 IFOA(3)<>0OROA(4)<>0OROA(5)<>0OROA(6)<>0THEN
10100
10085 S$="YOU HAVE NO GOODS LEFT*":GOSUB9100
10090 S$="FOR TRADING*":GOSUB9100
10095 GOTO10038
10100 S$="IN EXCHANGE FOR ANY KNIVES*":GOSUB9100
10102 S$="SALT CLOTH OR JEWELS YOU HAVE*":GOSUB910
0
10104 S$="HE OFFERS YOU PEARLS CARVINGS*":GOSUB910
0
10106 S$="AND SPICES*":GOSUB9100
10108 PRINT:GOSUB9200
10110 S$="WHEN YOU LEFT PORT THESE WERE*":GOSUB910
0
10112 S$="WORTH:*":GOSUB9100
10114 S$="PEARLS - 2 GLD PCS EACH*":GOSUB9100
10116 S$="CARVINGS - 2 GLD PCS EACH*":GOSUB9100
10118 S$="SPICES - 1 GLD PC PER GRAM*":GOSUB9100
10120 PRINT:GOSUB9200
10122 S$="BUT THEIR VALUE MAY HAVE*":GOSUB9100
10124 S$="CHANGED WHEN YOU GET HOME*":GOSUB9100
10125 PRINT:GOSUB9200:S$=K$:GOSUB9100
10126 GETI$:IFI$=""THEN10126
10130 FORT=3TO6
10135 IFOA(T)=0THEN10200
10140 PRINTCHR$(147):GOSUB9200
10145 PRINT"YOU HAVE":OA(T);
10150 IFT=3THENS$="BAGS OF SALT*"
10151 IFT=4THENS$="BALES OF CLOTH*"
10152 IFT=5THENS$="KNIVES*"
10153 IFT=6THENS$="JEWELS*"
10155 GOSUB9100
10156 PRINT:GOSUB9200
10160 S$="FOR THESE THE CHIEF OFFERS YOU*":GOSUB91
00
10165 PRINT"EITHER";OA(T)*EQ(T-2,1);"PEARLS"
10166 PRINT"    OR";OA(T)*EQ(T-2,2);"CARVINGS"
10167 PRINT"    OR";OA(T)*EQ(T-2,3);"GRAMS OF SPIC
E"
10168 PRINT:GOSUB9200
10170 S$="DO YOU WANT PEARLS,CARVINGS*":GOSUB9100
10172 S$="OR SPICES?*":GOSUB9100
10174 S$="(ENTER 1,2 OR 3)*":GOSUB9100
10175 INPUTI$
10176 I=VAL(I$):IFI<1ORI>3THEN10174
10180 AO(I)=AO(I)+(OA(T)*EQ(T-2,I))
10190 PRINT:PRINT"THE ";T$(I);" ARE PUT ON THE SHI
P"
10192 S$=K$:GOSUB9100
10194 GETI$:IFI$=""THEN10194
10200 NEXT
10210 PRINT:PRINT:GOSUB9200
10215 S$="    END OF TRADING*":GOSUB9100
10216 PRINT:GOSUB9200
10218 S$="YOU HAVE OBTAINED:*":GOSUB9100
10220 PRINTAO(1);"PEARLS"
10222 PRINTAO(2);"CARVINGS"
10224 PRINTAO(3);"GRAMS OF SPICE"
10226 PRINT:GOSUB9200
10228 S$=K$:GOSUB9100
10229 GETI$:IFI$=""THEN10229
10230 RETURN
```

Basic Flavours

Spectrum:

Replace EQ(,) by Q(,) and AO() by E() throughout and make the following changes:

```
10070 CLS
10126 LET I$=INKEY$:IF I$="" THEN GO TO 10126
10140 CLS:GO SUB 9200
10229 LET I$=INKEY$:IF I$="" THEN GO TO 10299
```

BBC Micro:

Make the following changes:

```
10070 CLS
10126 I$=GET$
10140 CLS:GOSUB 9200
10229 I$=GET$
```

WATTLEWORTH SILK McCONACHIE IN ASSOCIATION WITH CHRIS TUCKER

Grand Master
Most intelligent chess programs have the ability to 'look-ahead' a number of moves in the game to assess which of the next possible moves is the best. Computers can do this by constructing and searching a game tree. In the early days of AI research, the ability of a computer to play chess was considered to be the ultimate measure of machine intelligence

STRATEGIC PLANNING

The fundamental concepts behind chess-playing programs generally involve search strategies such as those we covered on page 1241, especially tree-search methods. As we continue our examination of AI, we delve into the mechanics of programs that can 'look ahead' and determine the best possible moves in games requiring strategy.

The phrase 'computer game' conjures up an image of zapping aliens in deep space or troll-bashing in subterranean caverns — but it was not always so. In the early days of computing, some of the pioneers of computer science — including such notables as Claude Shannon, John von Neumann and Alan Turing — devoted their efforts to programming a computer to play chess.

Chess was viewed as the intellectual game *par excellence,* and a successful computer chess program was regarded as the supreme test of a machine's intelligence. Today there are computer systems, such as Belle and the Cray Blitz, which play at the level of international masters, though few would claim that these machines think. Even so, chess and other games of mental skill provide the ideal arena for testing theories of strategic planning — open competition.

Most programs that play games of skill rely on tree-searching techniques, rather like those described on page 1241, but modified to take into account an adversary. The fundamental idea is that of 'look-ahead'. The program builds a game tree by considering its own moves, looking at the counter-moves available to the opponent, looking ahead to its responses to those and so on.

The game tree diagram shows the look-ahead tree for an imaginary two-person game. The root of the tree is the current position, with MAX ready to move. The terminal nodes, or leaves, are endgame positions. The tree is used in selecting which move to make by a procedure known as 'minimaxing', which was first clearly stated in 1949 by Claude Shannon. It works by first assigning numeric values to the terminal nodes — say one for a win, zero for a draw and negative one for a loss. These values then combine as we work up the tree, on the assumption that the player (MAX) always picks the largest while the opponent (MINI) always chooses the smallest, to produce values for higher nodes.

In this example, the value of the root is 0, indicating that the game will be drawn (provided that neither side makes a mistake). The correct move at the top level is thus M1, M3 or M4, but not M2. The rules that govern the branching and generation of node values are determined by the rules of the specific game. Only in trivial games, like noughts and crosses, is it possible to grow the entire game tree right to the end. Chess, for example, has a 'branching factor' of about 32 — meaning there are approximately 32 legal moves at any position. To look ahead 'four-ply' (two moves by each side) would lead to over a million terminal nodes. This 'combinatorial explosion' means that chess-playing programs cannot look ahead to the conclusion of the game.

Instead, most game programs look as far ahead as they can and evaluate the positions found there. To do so requires a method of assessing how favourable or unfavourable the leaf nodes are, even if the true result is not known. This is usually called the 'static evaluation function', and it necessarily introduces imprecision because it is only an estimate of the final outcome. Nevertheless, the rationale in looking some distance ahead and using an imperfect evaluation function is that it will be applied closer to the end of the game, and more likely be a better estimate than it would be with no search at all.

Taking draughts, or checkers, as an example, we might devise a very simple four-term evaluation function based on:

K King advantage
P Piece advantage
M Mobility difference

C Centre control

These attributes can be computed by examining the game board. For example, K = WK — BK where WK is the number of defending kings and BK represents the number of opposing kings.

The other features reflect several facts: it is better to have more pieces than your opponent (the loser ends up with no pieces); it is useful to have more moves available; and the central squares in checkers, as in chess, are more valuable than edge squares. The program must somehow combine these quantities into an overall score.

Assuming we decide that a king (K) is three times as valuable as an ordinary piece (P), an extra piece is worth two and a half additional moves (M) and a spare move is twice as good as controlling one more central square, our evaluation function would then be:

V = 15K + 5P + 2M + C

(Integer weightings are typically used for speed of calculation.)

This, however, is a very crude evaluation function. In contrast, Arthur Samuel's classic checker-playing program of the early 1960s employed up to 25 parameters. The coefficients here are rather arbitrary as well. Part of the fun in devising game-playing programs lies in tuning such weightings to achieve a good balance. One of the fine points of Samuel's program was that it adjusted its own weightings automatically, which was a rudimentary kind of learning.

The idea of giving numerical values to game features and combining them in a weighted sum to estimate the value of any position has proved its worth over the past 30 years. The evaluation function plays a similar role to that of the heuristic distance measure in problem-solving by search, discussed on page 1241.

A program that simply looks ahead a fixed depth and evaluates the terminal nodes that have been found will run into trouble. This is because some game positions are 'quiet' while others are highly 'unstable'. In chess, the state of play after a capture or pawn promotion is likely to be unstable: a recapture may take place on the very next move. If this takes place one ply beyond the program's 'horizon', the evaluation will be seriously misleading.

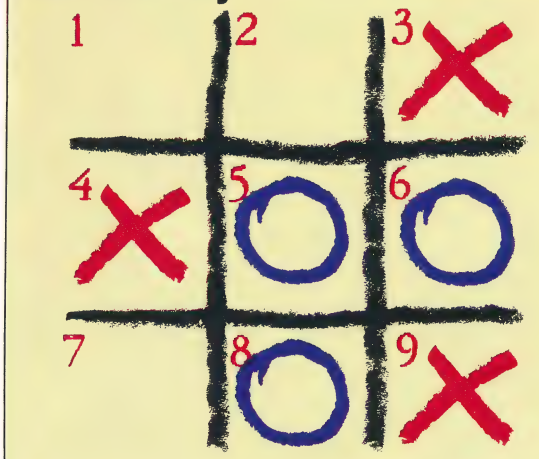
To reduce this effect, most programs do not have fixed-depth look-ahead. They have a measure of 'quiescence', which indicates whether a position may be reliably evaluated. If the position is not safe to evaluate, the search is pushed further ahead. In chess and checkers, this entails examining comparatively long capture sequences.

The alpha-beta algorithm first appeared in Greenblatt's MacHack program of 1967. It is a refinement of basic minimaxing and gives the same result but with far less effort. The diagram on the following page shows part of a game tree between two players called MINI and MAX.

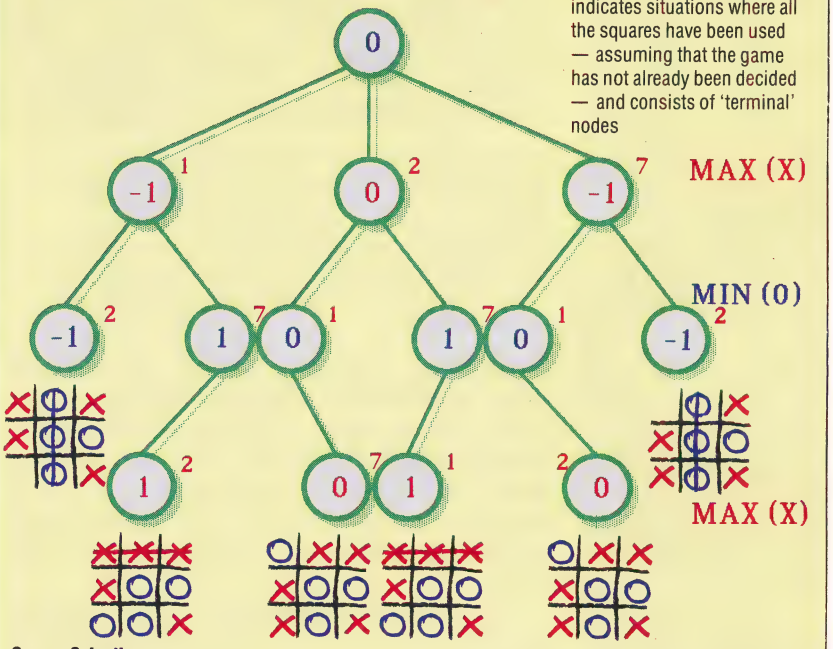
The letters against each node (A to L) show the order in which the tree is examined, using a depth-first procedure; the numbers are evaluations. The single strikes mark what are known as 'alpha cutoffs' and the double strikes denote 'beta cutoffs'. These prune away branches that cannot affect the end result.

An alpha cutoff occurs at node E, which never needs to be evaluated, and nor will any of its descendants, if it has any. We know by the time we reach node E that node C obtains a score of 15, but at node D the opponent can force us down to 10. There is no point in finding out if we can be forced even lower as this route is obviously less desirable than the route through node C. Thus node F's other descendants can be eliminated from consideration.

State Of Play



The diagram shows the current state of play in a game of noughts and crosses after six moves (with crosses to play next). A simple game tree can be constructed to trace the final three turns in the game by considering the number of options open at each stage. The first ply has three nodes, corresponding to the three possible moves (into squares 1, 2 or 7) that crosses can make. When it is noughts' turn to play, there are only two squares left to choose from, with three possibilities (of which two will be available after crosses has made a move). The next ply, therefore, has six nodes. The bottom ply indicates situations where all the squares have been used — assuming that the game has not already been decided — and consists of 'terminal' nodes



Square Selection

Having constructed the game tree, each terminal node can be assigned a value: 1 for a crosses win, 0 for a draw and -1 for a noughts win. We can then work back up the tree assigning values to the other nodes. If we consider the rightmost node in the first ply, the -1 value is arrived at by considering the two nodes below it, which have values 0 and -1. As the value is determined by noughts (it is noughts' turn to play) the minimum value will be selected — i.e. -1. Backing right up the tree to the current move, crosses should choose the maximum value of the three available. In this example, the best that crosses can do is to draw the game by selecting square 2 for the next move, the other two options will result in a noughts' win

LIZ DIXON

Making Moves

1	d2-d3	e7-e5
2	Nb1-d2	Nb8-c6
3	g2-g3	Ng8-f6
4	Bf1-g2	Bf8-c5
5	e2-e3	d7-d5
6	Ng1-e2	0-0
7	a2-a3	Bc8-f5
8	b2-b3	Nf6-g4?!
9	h2-h3	ng4-f6
10	Bc1-b2	Qd8-d6
11	g3-g4	Bf5-e6
12	Ne2-g3	a7-a5
13	Qd1-e2	Nf6-d7
14	Ng3-f5	Be6xf5
15	g4xf5	Nd7-f6
16	h3-h4	Rf8-e8
17	Bg2-h3	a5-a4
18	b3-b4	Bc5xb4!?
19	a3xb4	Qd6xb4
20	Bb2-a3	Qb4xh4
21	Nd2-f3	Qh4-h5
22	Nf3-d2	Qh5xe2+?
23	Ke1xe2	b7-b5
24	c2-c3	h7-h6
25	Bh3-g2	Ra8-a5
26	Ra1-b1	Re8-b8
27	Rb1-b2	Rb8-b6
28	Rh1-b1	Kg8-h7
29	Ba3-c5	Rb6-b8
30	Bc5-a3	Nc6-a7
31	Nd2-f3	Nf6-d7
32	Nf3-e1	c7-c6
33	Nd1-c2	Rb8-a8
34	Nc2-b4	Ra8-d8
35	Nb4-a2	Na7-c8
36	c3-c4!	d5xc4
37	d3xc4	b5xc4
38	Bg2xc6	Nc8-a7
39	Bc6-e4	Nd7-f6
40	Ba3-e7	Rd8-c8
41	Be7xf6	g7xf6
42	Rb1-b6!	c4-c3
43	Rb6xf6	Kh7-g7
44	Rf6-b6	a4-a3
45	Rb1-g1+	

At this point, the program indicated that it would be 2.4 pawns down, and the programmers resigned on its behalf

Position Of Play After Move 21

This chess game forms part of a series in which the world's most powerful computer chess program, the Cray Blitz, was beaten 4-0 by David Levy of Intelligent Software following a $5,000 wager between Mr Levy and the Blitz's programmers. Although the program has achieved a National Master chess rating, the game shows that a lot of work still needs to be done before computers can pose a serious challenge to the world's top human players

The same reasoning applies in reverse at node I. By the time we get there we know that G yields a score of 20. Node H, with 25, looks better, but MINI — not MAX — chooses between nodes G and J, and will clearly prefer G. There is therefore no need to see if I is even more inviting, since MAX will never be allowed to get there.

We can put these ideas in terms of a family tree. MAX is a male chauvinist who thinks that node C, for example, is the uncle of nodes D and E, which are both sons of the same father. MINI, on the other hand, is a feminist, and as far as she is concerned, G is the aunt of the sisters H and I, whose mother is J. As long as you are not troubled by nodes changing sex at alternative plies, this analogy will let us explain the alpha-beta rule quite concisely:

- As soon as MAX finds a son that is *worse* than any of his uncles, he ignores that son's remaining brothers.
- As soon as MINI finds a daughter that is *better* than any of her aunts, she ignores that daughter's remaining sisters.

At best, the alpha-beta algorithm only twice examines the square root of the number of tip nodes in the game tree, compared to simple minimaxing. At worst, it looks at just as many, and slightly more slowly. To avoid the first of our two cases, it is important to generate the brothers and sisters at each level in a sensible order. At maximizing levels they should be generated best-first and at minimizing levels worst-first (best-first for the opponent).

THE NUMBERS GAME

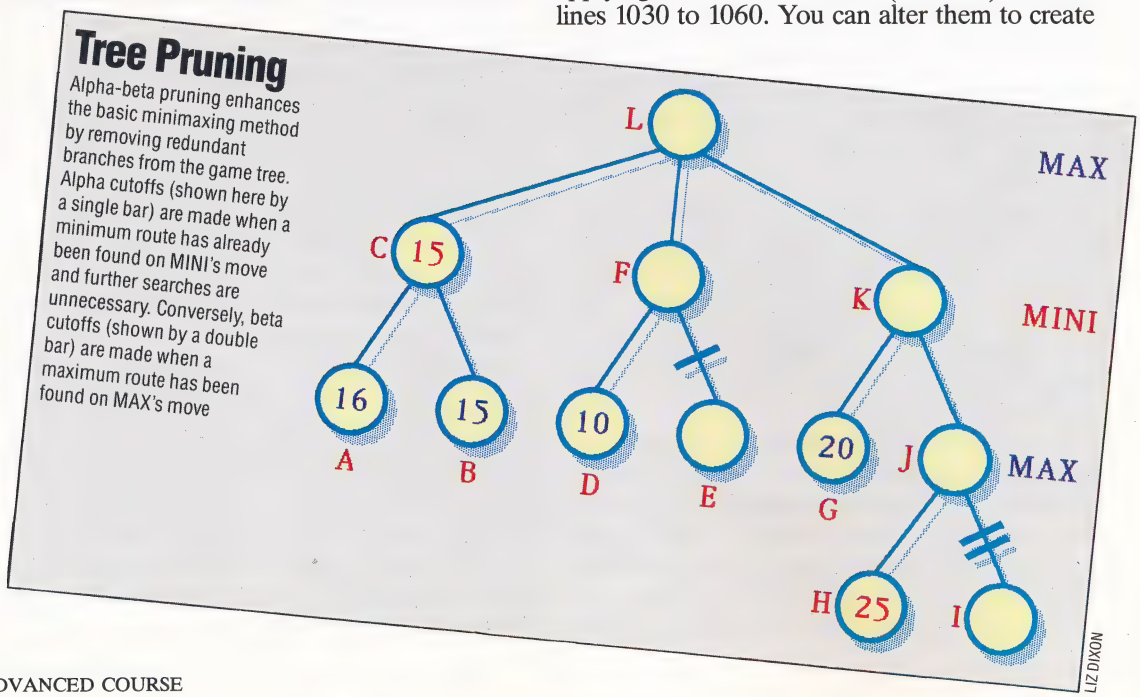
To illustrate the important concepts of tree-searching, we present an artificial game that comprises almost pure search techniques. This means that the details of board representation, move generation and static evaluation (which are crucial to the success of any real game program but which are also game-specific) do not obscure the essential simplicity of the alpha-beta procedure.

There is neither a board nor are there any pieces: the state of the game is described completely by a single number, held as V%. The object of the game for the computer is to raise the value of V% to 255, and for you to make it less than -255.

On each turn the player can choose between applying one of four functions (A, B, C, D) listed on lines 1030 to 1060. You can alter them to create

Tree Pruning

Alpha-beta pruning enhances the basic minimaxing method by removing redundant branches from the game tree. Alpha cutoffs (shown here by a single bar) are made when a minimum route has already been found on MINI's move and further searches are unnecessary. Conversely, beta cutoffs (shown by a double bar) are made when a maximum route has been found on MAX's move



LIZ DIXON

different versions of the game, such as making it more difficult for the computer.

The game is trivial but it illustrates the search strategy uncluttered by extraneous details. In addition, it is highly mathematical, so the computer has an in-built advantage. The alpha-beta minimaxing relies heavily, in this version, on the use of recursive functions with parameters and local variables. The parameters are as follows:

VV% Current game state
A% Alpha-best so far at this level
B% Beta-worst so far at this level
D% Depth indicator

As the program involves recursion, we include only a BBC BASIC version here. A non-recursive version for Commodore 64 and Spectrum owners will be given in the next instalment.

The Numbers Game

```
 10 REM ******************************
 11 REM ** Listing 3.1 :            **
 12 REM ** THE NUMBERS GAME         **
 13 REM ******************************
 50 MODE 7
100 REM -- Game to demonstrate search:
120 GOSUB 1000  : REM initialization
130 GOSUB 1600  : REM instructions
150 REPEAT
160   GOSUB 2000  : REM prepare new game
170   INPUT "Who goes 1st (1=You, 2=Me) ", H1%
180   IF H1%<1 OR H1%>2 THEN GOTO 170
200   REM -- main loop:
210   REPEAT
220     IF H1%=1 THEN GOSUB 3000
230     REM -- person's turn.
240     GOSUB 3500  : REM board display
250     H1%=1 : REM always 1 after 1st cycle
260     GOSUB 4000  : REM test for win
270     IF EG%=0 THEN GOSUB 5000
280     REM -- computer's turn.
290     GOSUB 3500  : REM show game status
300     GOSUB 4000  : REM test for endgame
310     UNTIL EG%<>0 OR M%>33
320   REM -- finale:
330   GOSUB 6000 : REM congratulations
340   PRINT "Another Game (N=No) ";
350   Y$=GET$
360   UNTIL Y$="N" OR Y$="n"
365 PRINT
370 PRINT "So Long, and thanks for the game!"
400 END
444 :
500 DEF FNmaximove(VV%,A%,B%,D%)
505 REM ---------- my turn:
510 LOCAL P%,E%,KEEP%
515 IF D%>=MD% OR ABS(VV%)>HI% THEN =VV% :
    REM static value
520 REM else go deeper:
525 P%=0
530 REPEAT  P%=P%+1
535   H%=P%: V%=VV%: GOSUB 5500: REM make move
536   IF D%=1 THEN PRINT CHR$(H%+64);" = ";
540   E%=FNminimove(V%,A%,B%,D%+1)
545   IF E%>A% THEN A%=E%: KEEP%=P%
548   IF D%=1 THEN PRINT E%;"; ";
550   UNTIL P%>3 OR A%>=B%
555 IF A%>BV% AND D%=1 THEN BV%=A%: HH%=KEEP%
556 REM keep best so far.
560 =A%
565 REM return with max A%
570 :
700 DEF FNminimove(VV%,A%,B%,D%)
710 REM ---------- other's turn:
720 LOCAL E%,P%
730 IF D%>=MD% OR ABS(VV%)>HI% THEN =VV%
740 P%=0
750 REPEAT P%=P%+1: H%=P%
755   V%=VV%: GOSUB 5500: REM make move
760   E%=FNmaximove(V%,A%,B%,D%+1)
770   IF E%<B% THEN B%=E%
780   UNTIL P%>3 OR B%<=A%
790 =B%
796 REM returns with lowest value B%
999 :
1000 REM -- initializing routine:
1001 BL$=""
1002 @%=4
1020 REM -- the 4 functions:
1030 DEF FNA(X%)=2*X% - 7
1040 DEF FNB(X%)=X% DIV 2 + 1
1050 DEF FNC(X%)=-4*X% + 17
1060 DEF FND(X%)=3*X% - 4
1070 LO%=-255: HI%=255
1150 RETURN
1160
1600 REM -- instruction routine:
1610 CLS: PRINT
1620 PRINT "Welcome To The Numbers Game!"
1630 PRINT "If you don't know the rules,"
1635 PRINT "READ THE ARTICLE!"'
1636 PRINT "N.B. I maximize : you minimize."
1640 PRINT "To see the effect of a move type:"
1645 PRINT "A, B, C, or D. To make it, type X."
1650 PRINT : PRINT "Good Luck!";CHR$(7)
1660 RETURN
1670
2000 REM -- Preparation routine:
2010 M%=0 : REM moves
2020 V% = RND(15)-8 : REM initial state.
2050 EG%=0
2060 PRINT'"Initial state = ";V%
2100 RETURN
2110
3000 REM -- Person's Move
3010 M%=M%+1
3020 PRINT
3030 REPEAT
3040   PRINT "Your move is ? ";
3050   H$=GET$: PRINT H$;
3060   IF H$="A" THEN PRINT FNA(V%): H%=1
3070   IF H$="B" THEN PRINT FNB(V%): H%=2
3080   IF H$="C" THEN PRINT FNC(V%): H%=3
3090   IF H$="D" THEN PRINT FND(V%): H%=4
3100   UNTIL H$="X"
3120 GOSUB 5500 : REM do choice H%
3150 RETURN
3160 :
3500 REM -- board display routine:
3520 CLS : PRINT
3522 PRINT "     Move ";M%;" --> ";
3523 IF M%<1 THEN RETURN
3525 PRINT CHR$(64+H%);
3530 PRINT " = ";V%
3535 RETURN
3700
4000 REM -- Win-test Routine (on M$):
4001 IF M%<1 THEN RETURN
4010 EG%=0
4020 IF V%<LO% THEN EG%=-1
4030 IF V%>HI% THEN EG%=1
4040 RETURN
4050 :
5000 REM -- Computer's Move Routine:
5005 W%=V% : REM save current status.
5010 M%=M%+1
5015 MD%=6 : REM max depth
5020 IF M%<4 THEN MD%=4
5030 IF M%>8 THEN MD%=8
5040 GOSUB 5200 : REM --> H%
5045 V%=W% : REM restore status.
5050 GOSUB 5500 : REM do it.
5070 RETURN
5080 :
5200 REM -- Move Selection:
5210 BV%=LO% : D%=0
5220 BV%=FNmaximove(V%,LO%,HI%,1)
5230 H%=HH%
5240 PRINT'"Press any key to go on: ";
5244 C%=GET
5250 RETURN
5270 :
5500 REM -- Make-move Routine (H% : V%):
5505 ON H% GOTO 5510,5520,5530,5540
5510 V%=FNA(V%): RETURN
5520 V%=FNB(V%): RETURN
5530 V%=FNC(V%): RETURN
5540 V%=FND(V%): RETURN
5550 :
6000 REM -- Congratulations Routine:
6010 PRINT "GAME OVER!"
6020 IF EG%>0 THEN PRINT "I won it!!"
6030 IF EG%<0 THEN PRINT "Well done!"
6040 IF EG%=0 THEN PRINT "Game drawn"
6050 RETURN
6600 :
```

THE PROLOGUE

First-order predicate logic, which is used in PROLOG, is very similar to the way in which we solve most everyday problems and therefore makes the language both useful and friendly. We begin here a series on PROLOG by outlining the structure of the logic we'll be dealing with, as well as looking at the origins of the language.

It seems a long time ago since the Japanese declared they would change the course of computer technology by pouring money and research effort into their Fifth Generation computer project. Perhaps one of the most significant decisions they made at the outset was to use the little-known programming language PROLOG as the 'core language' of the high-performance, intelligent database machines that they foresee.

PROLOG is an acronym for '*pro*gramming in *log*ic' and is a good but imperfect realisation of that ideal. But why is it desirable to program in logic at all? There are many logics that can be used to describe the world and aspects of it. Some you will be familiar with, such as mathematics, while others can seem quite esoteric, like certain philosophical doctrines. First-order predicate calculus is a logic rather like the one we use in everyday thought and discussion, although it has its own notation and certain restrictions. A predicate is best thought of as a relationship between things. In the sentence 'John likes Anna', the predicate is 'likes'. We could write this as 'likes (John, Anna)', which is less readable but clearer as to which is the predicate and which are its arguments. To say that John is male, we could write 'male(John)', where 'male' is a predicate taking one argument ('John'). Similarly, 'female(Anna)' means that Anna is female.

What we have here are simply declarations of fact, but we can extend the logic to show how some facts *imply* others. Using predicate logic, we can describe the world in terms of facts and implications and use our description to deduce new facts from the old ones. We do this by introducing variables. Logic variables are much the same as those you may be familiar with in BASIC or some other programming languages, except their scope is limited to the clause (fact or implication) in which they appear, rather than the whole collection of clauses. This means that the John who likes Anna could possibly be a different John to the one who is male.

We could now write a fact such as 'female(X) → likes (John, X)'. The arrow means 'implies' so we can read this as a rule that states 'the fact that X is female implies that John likes X'. In more usual English, the rule reads 'John likes X if X is female'. This rule is an example of a particular kind of clause in the predicate calculus called a *horn clause*. Horn clauses have a statement as their head (the *consequent*), which is true only if all the statements in the body (the *antecedents*) are true.

A if B and C and D

is a horn clause with the head A and the body B, C and D. A simple fact can be seen as a consequent with no antecedents and is assumed to be true.

PROGRAMMING IN LOGIC

With logic we can express the program we wish to write as a collection of facts and rules that describe the things we are interested in. This description is in a form fairly similar to the way in which we actually conceive the problem. To 'run' our logic program we try to prove the truth or falsity of some statement. If the statement is a simple fact then we can assume it is true without further effort. If it is the consequent of some rule, then we must set about proving the truth of all its antecedents before we can say it is true. So, if we want to know whether John is male, we try to prove the statement 'male(John)', which we know is true because it is a fact we already have. If we want to know whether Anna likes John, we need to first prove the statements 'male(John)' and 'likes(John,Anna)'.

PROLOG was developed at the University of Marseilles in the early 1970s by A. Colmerauer, based partly on the work of Bob Kowalski who is now at Imperial College, London. It

LIZ HEANEY

Prolog Flavours Prelude

PROLOG tends to require generous amounts of RAM in order to be efficiently implemented, and versions of the language that will run on a home computer are rare. Spectrum owners however, can consider purchasing MICRO-PROLOG, written by Logic Programming Associates and distributed by Sinclair on cassette for £24.95.

Owners of other micros can take heart from the fact that PROLOG is rapidly attracting attention from software producers and versions of the language are currently planned for the BBC Micro and the Enterprise.

Spectrum MICRO-PROLOG differs in many respects from standard DEC10 PROLOG, the version used for examples in our series. We will, however, be printing a series of short PROLOG Flavours to enable Spectrum owners to enter listings directly with MICRO-PROLOG. The most significant differences will be explained later in the series

uses only the horn clauses of predicate logic and a notation similar to the one we have just shown. The whole of the predicate calculus could have been implemented, but PROLOG, like any programming language, is a compromise between computational efficiency and expressive power. Much research is being done to fine-tune this balance, although PROLOG in its present form appears quite stable.

The nearest thing to a standard PROLOG is a version known as DEC-10 PROLOG (because it was first implemented on a Digital Equipment Corporation mainframe) and the PROLOG user's bible is a book by Clocksin and Mellish called, simply enough, *Programming in PROLOG*, which describes this standard and gives much practical detail besides. Almost all current PROLOGS , including C-PROLOG, are modelled on this standard, although flavours and dialects abound. For the micro, there are two front-runners: the version from Expert Systems, which is very close to the standard, and MICRO-PROLOG from Logic Programming Associates which, although popular, differs considerably in its syntax and internal structure.

PROLOG implementations use a lot of memory and for this reason do not fit very comfortably into anything much smaller than a micro with 64 Kbytes of RAM. However, MICRO-PROLOG is available for the Spectrum and new implementations continue to appear.

PROLOG programs do not exhibit the familiar flow of control in which the first program statement is executed, then the second and so on until the last, with the odd branch and loop along the way. Instead, PROLOG uses a technique known as 'backtracking'.

In order to solve a query, PROLOG works its way down through a chain of rules, setting itself a new goal to prove each time. If a particular path through the chain proves unproductive, PROLOG will 'backtrack' to an earlier choice point and then branch off in a new direction. We will be looking at this in greater detail later in the series.

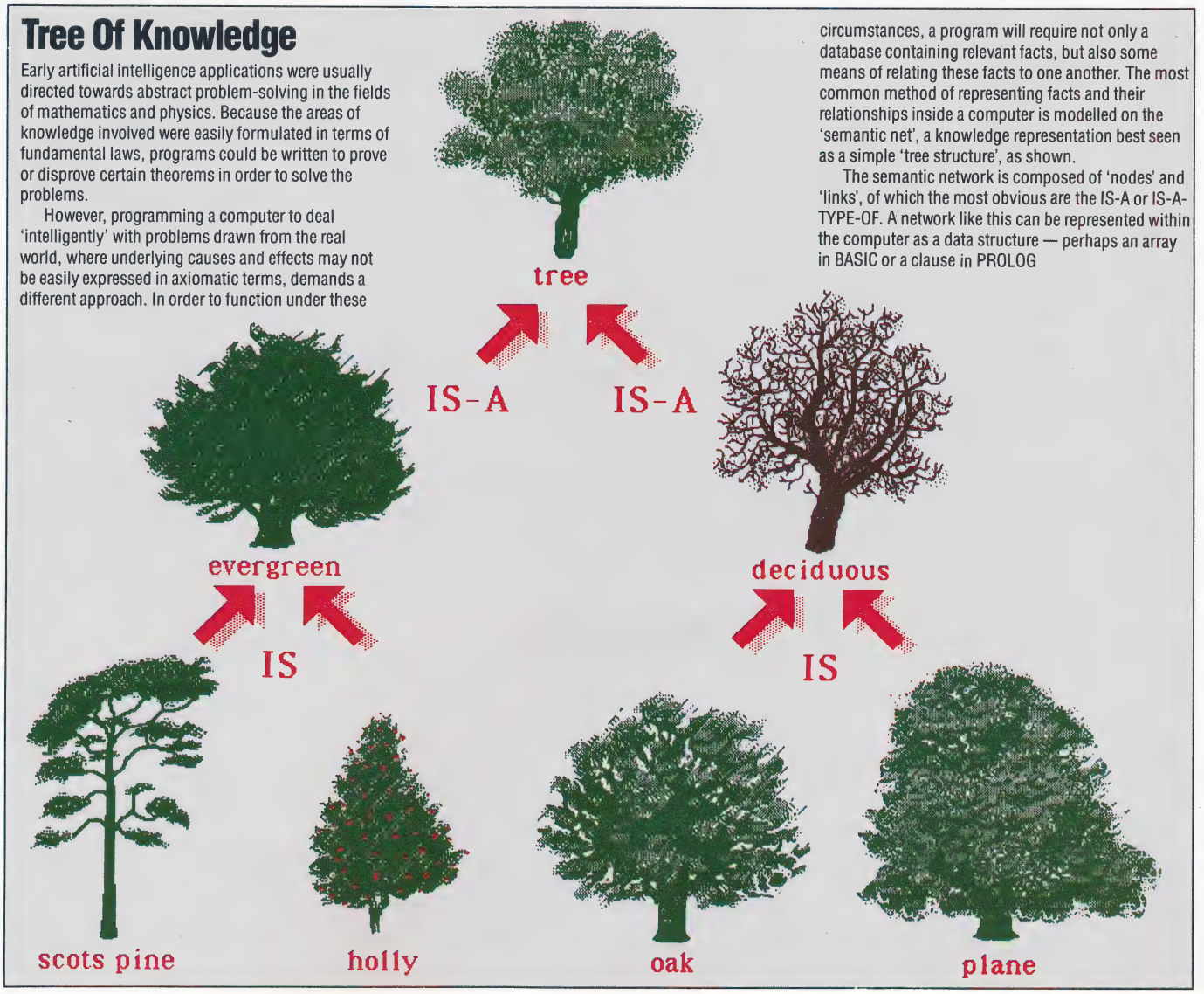
This way of proceeding gives PROLOG a very different 'feel' compared to any other programming language. Proponents of logic programming stress the declarative nature of a PROLOG program. That is, the reading of a rule as though it were a clause in first order predicate calculus — for example, X is an uncle of Y if X is male and X is the sibling of Z and Z is the parent of Y. Yet PROLOG can always be read in the more familiar procedural style (to prove that X is an uncle of Y, first show that X is male, then show that X is the sibling of Z, and then show that Z is the parent of Y). The availability of the declarative reading is something that almost all other languages lack and it is certainly a very valuable feature for helping you understand and therefore design and test your programs.

Tree Of Knowledge

Early artificial intelligence applications were usually directed towards abstract problem-solving in the fields of mathematics and physics. Because the areas of knowledge involved were easily formulated in terms of fundamental laws, programs could be written to prove or disprove certain theorems in order to solve the problems.

However, programming a computer to deal 'intelligently' with problems drawn from the real world, where underlying causes and effects may not be easily expressed in axiomatic terms, demands a different approach. In order to function under these circumstances, a program will require not only a database containing relevant facts, but also some means of relating these facts to one another. The most common method of representing facts and their relationships inside a computer is modelled on the 'semantic net', a knowledge representation best seen as a simple 'tree structure', as shown.

The semantic network is composed of 'nodes' and 'links', of which the most obvious are the IS-A or IS-A-TYPE-OF. A network like this can be represented within the computer as a data structure — perhaps an array in BASIC or a clause in PROLOG



IAN McKINNELL

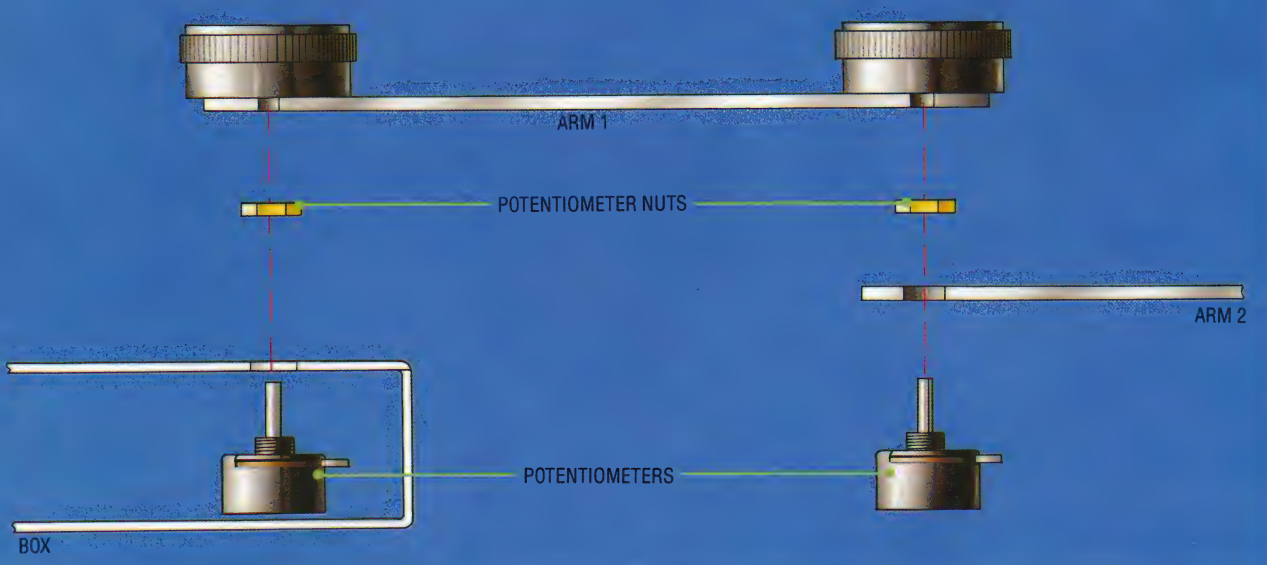
SETTING YOUR SIGHTS

We continue our digital tracer project by assembling the arm components and adding the potentiometers and tracer sight. This will complete the constructional phase of the project, leaving us the calibration and programming, which we will present in the next instalment.

Step 1: Arm Assembly

The two arm pieces we cut out in the last instalment are hinged by a pair of potentiometers, one of which is mounted in the plastic component box lid and the other through the connecting arm ends. Begin by cutting the spindle of each potentiometer so that 15mm only protrudes from the body. Fix one potentiometer into the box lid and the other into Arm 2, fastening in place with the potentiometer nuts.

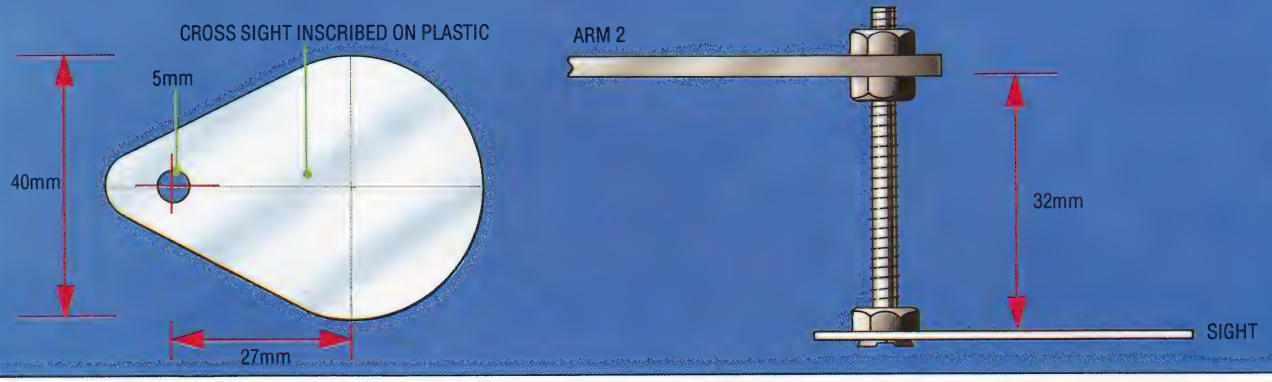
Note that the box potentiometer connection tags should point towards the nearest box side, while the other potentiometer's tags should point along the arm as shown. Push the potentiometer spindles into the relevant knobs, already mounted on Arm 1, and fix in place by tightening the grub screws. Each spindle should be positioned so that the flat edge is clamped by the grub screw.

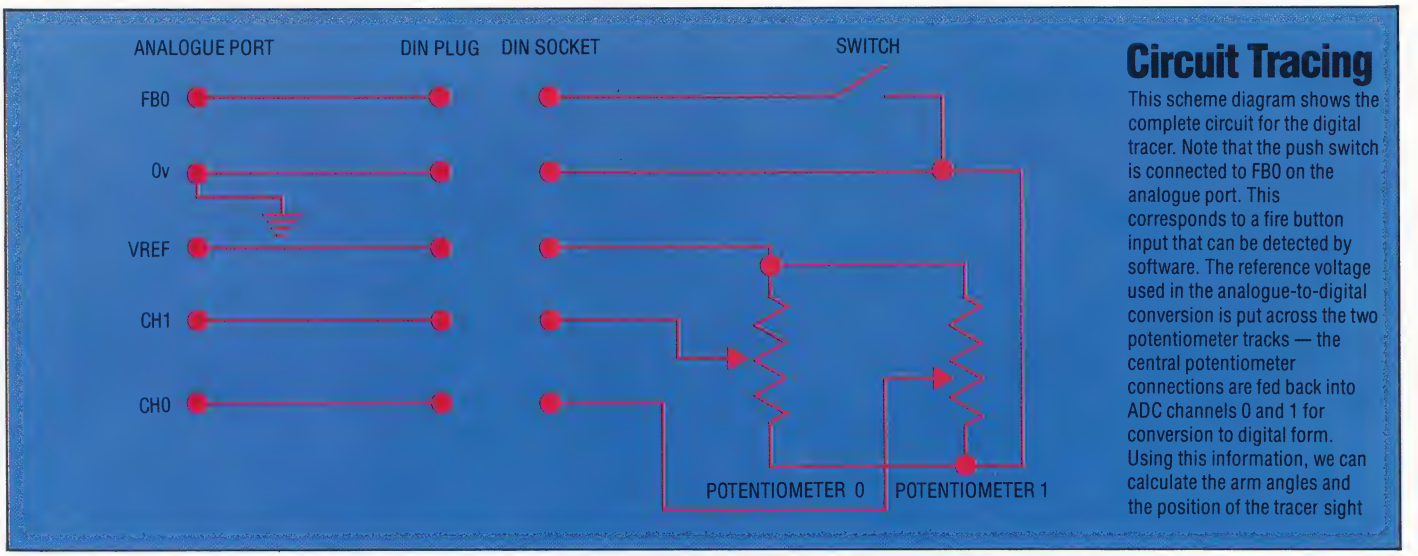


Step 2: Mounting The Sight

Cut a sight from a piece of a relatively thin transparent plastic, such as that used for audio cassette library cases. The exact shape of the sight is not important but the cross hairs should be etched onto the surface so that they cross as shown and are 27mm away from the centre of the mounting hole.

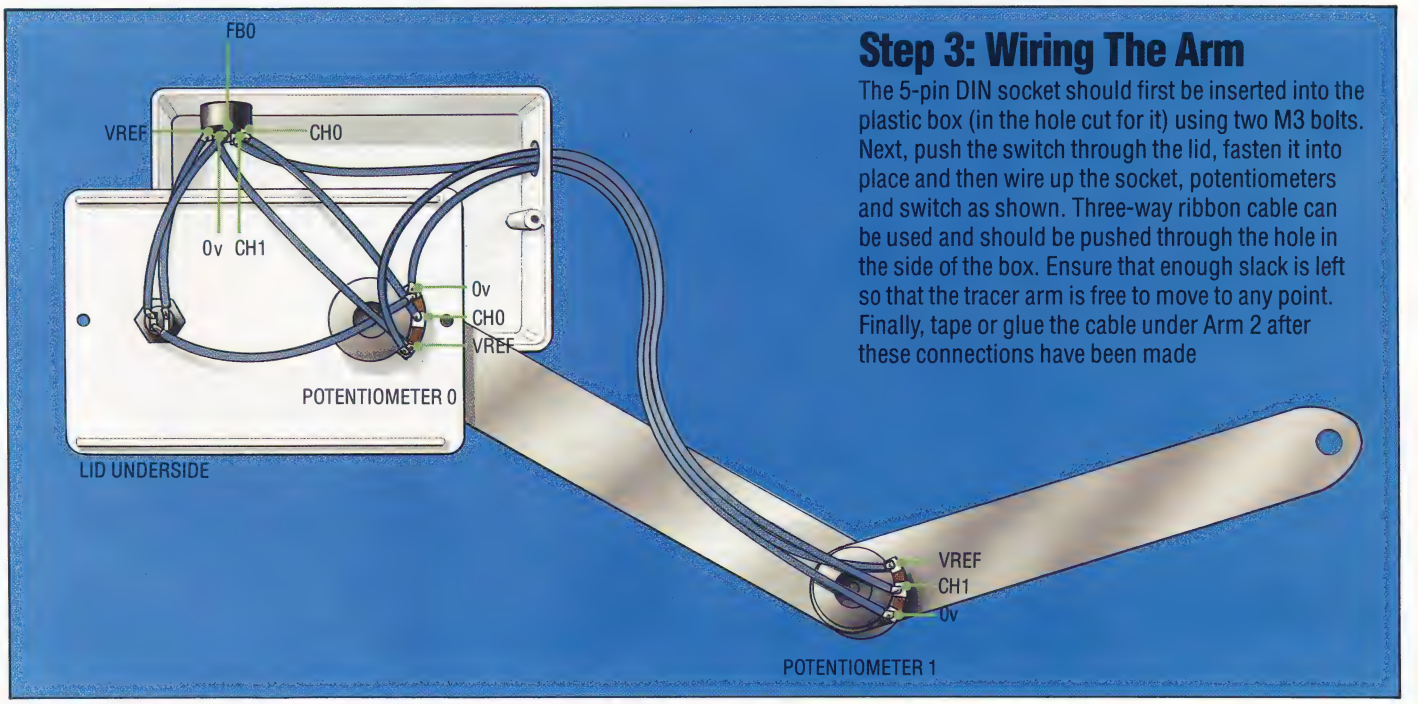
Attach the sight to the free end of Arm 2 with the M5 bolt and three nuts. Set the initial height to 32mm. This arrangement will allow the height to be easily changed





Circuit Tracing

This scheme diagram shows the complete circuit for the digital tracer. Note that the push switch is connected to FB0 on the analogue port. This corresponds to a fire button input that can be detected by software. The reference voltage used in the analogue-to-digital conversion is put across the two potentiometer tracks — the central potentiometer connections are fed back into ADC channels 0 and 1 for conversion to digital form. Using this information, we can calculate the arm angles and the position of the tracer sight

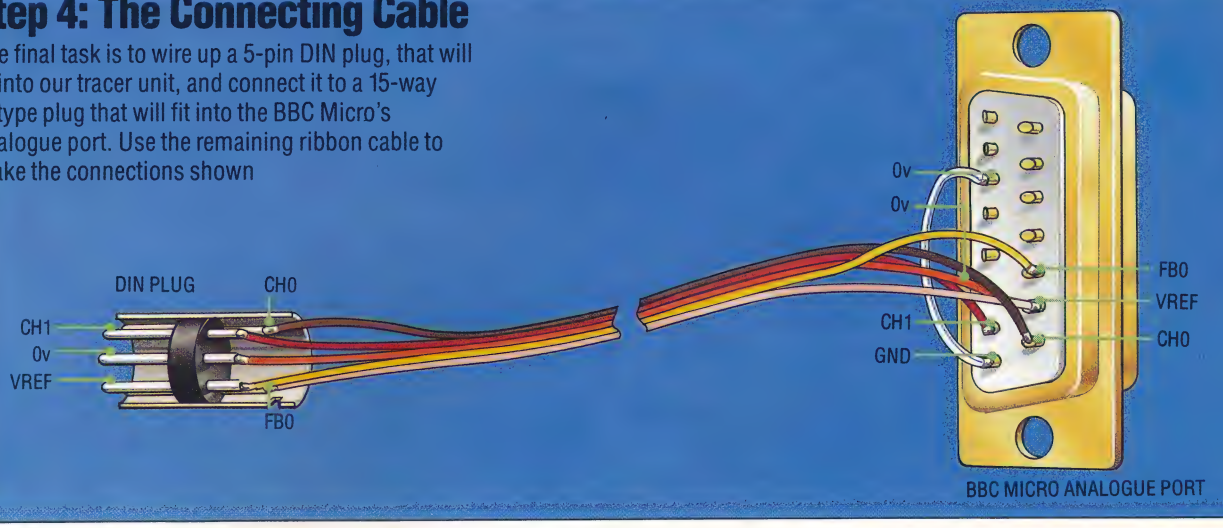


Step 3: Wiring The Arm

The 5-pin DIN socket should first be inserted into the plastic box (in the hole cut for it) using two M3 bolts. Next, push the switch through the lid, fasten it into place and then wire up the socket, potentiometers and switch as shown. Three-way ribbon cable can be used and should be pushed through the hole in the side of the box. Ensure that enough slack is left so that the tracer arm is free to move to any point. Finally, tape or glue the cable under Arm 2 after these connections have been made

Step 4: The Connecting Cable

The final task is to wire up a 5-pin DIN plug, that will fit into our tracer unit, and connect it to a 15-way D-type plug that will fit into the BBC Micro's analogue port. Use the remaining ribbon cable to make the connections shown



KEVIN JONES

GRAPHICS REVOLUTION

In the first of these two articles dealing with 3-D graphics on the Commodore 64, we defined our machine code programming objective using a BASIC program. In this instalment, we complete the conversion of the BASIC listing and introduce more interpreter floating point routines.

The hybrid program (Test I-Rot and I-Rot.Hex) developed on page 1256 runs quite quickly. However, it is apparent that scanning through the adjacency matrix E%(I,J)(which defines the nodes that are connected in the frame figure) to discover which points are to be plotted is slowing the program up. To speed things up further, therefore, we need to code the remainder of the key BASIC loop of the Rotating Cube program (see page 1259) into machine code and run the entire loop in machine code. The result is a satisfying gain in speed.

In order to handle the following mathematical calculations:

```
X1%=X(I)+159:Y1%=199-(Z(I)+100)
X2%=X(J)+159:Y2%=199-(Z(J)+100)
```

found in lines 1640 and 1650 of the original BASIC program, we will need some further interpreter calls. Essentially, both of these lines of BASIC take a floating point variable — X(I), say — and add 159 (in floating point format) before taking the integer part and storing it in two-byte format as X1%.

The interpreter calls needed to accomplish this are as follows:

- FLPINT (call address $B1AA):

This routine takes the integer part of the number in FAC and returns the result (if in the range -32767 to 32767) in lo/hi byte format in the Y and A registers, respectively. Note the unusual lo/hi order here, which is the reverse of most other interpreter routines.

- SNGFT (call address $B3A2):

This routine takes a single-byte integer (in the range 0 to 255) from the Y register and places it in FAC in floating point format.

SNGFT is used in the subroutine SETUP of the Assembly listing (line 5150). For example, the decimal value 159 is placed in the Y index register and SNGFT is called to convert it and place the result in FAC. After this, MOVMF is used to place the result in the five bytes of MEM1. Thus, when we want to add 159, it is available for use in MEM1.

The main problems remaining in converting the BASIC loop to machine code concern specifying particular elements in the arrays defining the shape to be rotated. Calculation of array pointers can prove difficult in certain circumstances. The co-ordinate arrays X(I),Y(I),Z(I) pose no special difficulties, since in each array we simply add five bytes to the pointer to get the address of the next element.

The array E%(I,J) however, being two-dimensional, is entirely different. It is arranged consecutively in memory as:

```
E%(0,0)E%(1,0)E%(2,0)...E%(NP,0)
E%(0,1)E%(1,1)E%(2,1)...E%(NP,1) and so on...
```

In other words, the array consists of blocks of memory, each 2×(NP+1) bytes long — each block corresponding to values of the second subscript and each element taking two bytes (as the array is an integer array).

Our plan is to mirror the BASIC code as exactly as possible in machine code so the I,J loops — which scan E%(I,J) — will go:

```
FORI=1TONP
FORJ=1TOI
```

This means that the change in the pointer to accomplish a machine code equivalent of NEXT I, allowing for the fact we want to skip any elements with a first subscript of zero, becomes quite complicated. To perform the rotation of the figure, the elements of E%(I,J) should be accessed in the following order:

```
E%(1,1)
E%(1,2)E%(2,2)
E%(1,3)E%(2,3)E%(3,3)
E%(1,4)E%(2,4)E%(3,4)E%(4,4) etc.
```

A quick calculation shows that 2×(NP+1) must be added to the pointer each time I is incremented. The best way to implement this in 6502 machine code is to use indirect addressing to access E%(I,J). The code for this is:

```
LDY JINDEX
LDA (ZPTEMP),Y
```

where ZPTEMP is a zero page, two-byte pointer, and JINDEX is used to keep track of J. ZPTEMP must also be incremented on each J increment. By incrementing ZPTEMP and the Y register, the net offset is increased by the necessary two bytes for each increment of J. The end result of these considerations is that ZPTEMP must be incremented by: (2×NP+1)-(I-1) on each iteration of the I loop. (I-1) is subtracted from the block length because ZPTEMP has already been incremented (I-1) times on the (just completed) J loop. Using this expression to calculate the offset means that ZPTEMP points to the correct byte after I is incremented.

Finally, it would be helpful if we could call an interpreter routine to locate the variable E%(I,J) for us. One such routine does exist, but unfortunately it is tortuous (it deals with all possible variable types) and extremely slow, so we must do the computation of the offset from the address of E%(1,1) ourselves.

Rotating Routines

The following listings give the source code for a machine code program that rotates a wire figure, defined by the array E%(I,J), on the high resolution screen. The program uses the interpreter routines and methods of locating BASIC variables discussed in the main text. Our initial aim of converting the original BASIC loop that scans the arrays to produce the rotating figure is now achieved. Also included are a BASIC Loader for this source code, together with a BASIC test program that sets up the machine code variables and calls the routine. The test program can be used with an assembled version of the source code, II-ROT.HEX, or by loading and running the Loader program, typing NEW and finally loading and running the test program. In this second case, line 1030 of the test program should be omitted.

BASIC Loader

```
1000 REM**  INSERT M/C II-ROT.HEX    **
1010 REM*****************************
1020 DATA72,138,72,152,72,32,101,197
1030 DATA173,83,197,172,84,197,32,162
1040 DATA187,173,75,197,172,76,197,32
1050 DATA40,186,162,94,160,197,32,212
1060 DATA187,173,87,197,172,88,197,32
1070 DATA162,187,173,77,197,172,78,197
1080 DATA32,40,186,169,94,160,197,32,86
1090 DATA184,162,94,160,197,32,212,187
1100 DATA173,87,197,172,88,197,32,162
1110 DATA187,173,75,197,172,76,197,32
1120 DATA40,186,162,99,160,197,32,212
1130 DATA187,173,83,197,172,84,197,32
1140 DATA162,187,173,77,197,172,78,197
1150 DATA32,40,186,169,99,160,197,32
1160 DATA103,184,162,99,160,197,32,212
1170 DATA187,169,94,160,197,32,162,187
1180 DATA174,83,197,172,84,197,32,212
1190 DATA187,165,88,160,197,32,162,187
1200 DATA174,87,197,172,88,197,32,212
1210 DATA187,240,3,76,197,240,31,169,5,24
1220 DATA109,83,197,141,83,197,144,3
1230 DATA238,84,197,169,5,24,109,87,197
1240 DATA141,87,197,144,3,238,88,197,76
1250 DATA112,197,169,1,141,0,193,141,1
1260 DATA193,141,2,193,32,14,193,32,101
1270 DATA199,169,1,141,81,197,141,82
1280 DATA197,173,79,197,133,253,173,80
1290 DATA197,133,254,172,82,197,177,253
1300 DATA208,3,76,207,198,173,83,197
1310 DATA174,84,197,32,162,187,169,94
1320 DATA160,197,32,103,184,32,170,177
1330 DATA140,0,195,141,1,195,73,85,197
1340 DATA172,86,197,32,162,187,169,94
1350 DATA160,197,32,103,184,32,170,177
1360 DATA140,2,195,141,3,195,73,89,197
1370 DATA172,90,197,32,162,187,169,99
1380 DATA160,197,32,80,184,32,170,177
1390 DATA140,4,195,173,91,197,172,92
1400 DATA197,32,162,187,169,99,160,197
1410 DATA32,80,184,32,170,177,140,5,195
1420 DATA173,0,195,205,2,195,208,19,173
1430 DATA1,195,205,3,195,208,11,173,4
1440 DATA195,205,5,195,208,3,76,207,198
1450 DATA32,14,195,173,81,197,205,82
1460 DATA197,240,40,169,5,24,109,85,197
1470 DATA141,85,197,144,3,238,86,197
1480 DATA169,5,24,109,91,197,141,91,197
1490 DATA144,3,238,92,197,230,253,208,2
1500 DATA230,254,238,82,197,76,73,198
1510 DATA173,74,197,205,81,197,240,88
1520 DATA169,5,24,109,83,197,141,89,197
1530 DATA144,3,238,84,197,169,5,24,109
1540 DATA89,197,141,89,197,144,3,238,90
1550 DATA197,169,1,24,109,74,197,10,56
1560 DATA237,81,197,24,105,1,134,101,252
1570 DATA133,253,165,254,105,0,133,254
1580 DATA173,68,197,141,85,197,173,69
1590 DATA197,141,86,197,173,72,197,141
1600 DATA91,197,173,73,197,141,92,197
1610 DATA169,1,141,82,197,238,81,197,76
1620 DATA73,198,104,168,104,170,104,96
1630 DATA173,68,197,141,83,197,141,85
1640 DATA197,173,69,197,141,84,197,141
1650 DATA86,197,173,70,197,141,87,197
1660 DATA173,71,197,141,88,197,173,74
1670 DATA197,141,93,197,173,72,197,141
1680 DATA89,197,141,91,197,173,73,197
1690 DATA141,90,197,141,92,197,160,159
1700 DATA32,162,179,162,94,160,197,32
1710 DATA212,187,160,99,32,162,179,162
1720 DATA99,160,197,32,212,187,96
1730 DATA79160:REM*CHECKSUM*
1740 FORI=50536TO51123
1750 READX:POKEI,X:CC=CC+X
1760 NEXT
1770 READX:IFX<>CCTHENPRINT"CHECKSUM ERROR":STOP
1780 PRINT"II-ROT.HEX INSTALLED OK"
```

Assembly Listing

```
1010 ;++++++++++++++++++++++++++++
1020 ;++      ROTSUB 64         ++
1030 ;++                        ++
1040 ;++++++++++++++++++++++++++++
1140 ;
1150 *          = $C544
1190 ; VARIABLES CALLED FROM BASIC
1200 ;
1210 XBASLO   *=*+1 ; POKE50500,X(1)LO
1220 XBASHI   *=*+1 ; POKE50501,X(1)HI
1230 YBASLO   *=*+1 ; POKE50502,Y(1)LO
1240 YBASHI   *=*+1 ; POKE50503,Y(1)HI
1250 ZBASLO   *=*+1 ; POKE50504,Z(1)LO
1260 ZBASHI   *=*+1 ; POKE50505,Z(1)HI
1270 NP       *=*+1 ; POKE50506,NP
1280 CSLO     *=*+1 ; POKE50507,CSLO
1290 CSHI     *=*+1 ; POKE50508,CSHI
1300 SNLO     *=*+1 ; POKE50509,SNLO
1310 SNHI     *=*+1 ; POKE50510,SNHI
1320 EBASLO   *=*+1 ; POKE50511,E%(1,1)LO
1330 EBASHI   *=*+1 ; POKE50512,E%(1,1)HI
1340 ;
1350 ; VARIABLES USED BY M/C
1370 IINDEX   *=*+1
1380 JINDEX   *=*+1
1390 XILO     *=*+1
1400 XIHI     *=*+1
1410 XJLO     *=*+1
1420 XJHI     *=*+1
1430 YILO     *=*+1
1440 YIHI     *=*+1
1450 ZILO     *=*+1
1460 ZIHI     *=*+1
1470 ZJLO     *=*+1
1480 ZJHI     *=*+1
1490 TEMPNP   *=*+1
1500 MEM1     *=*+5 ;FLOATING POINT VAR
1510 MEM2     *=*+5 ;FLOATING POINT VAR
1520 ZEROTEMP   =$FD ;FREE ZERO PAGE LOC
1530 ;
1540 ;INTERPRETER ARITHMETIC CALLS
1560 FMULT      =$BA28 ; FAC=FAC*MEM
1570 FSUB       =$B850 ; FAC=MEM-FAC
1580 FADD       =$B867 ; FAC=FAC+MEM
1590 MOVFM      =$BBA2 ; FAC=MEM
1600 MOVMF      =$BBD4 ; MEM=FAC
1610 FLPINT     =$B1AA ; .Y/.A=INT(FAC) N.B. HI/LO ORDER !!
1620 SNGFT      =$B3A2 ; FAC= .Y
1630 ;
1640 ;OTHER MACHINE CODE ROUTINE/VARS
1660 LINSUB     =$C30E
1670 MLO        =$C300
1680 MHI        =$C301
1690 NLO        =$C302
1700 NHI        =$C303
1710 Y1         =$C304
1720 Y2         =$C305
1740 ;++++ SAVE REGISTERS++
1760          PHA
1770          TXA
1780          PHA
1790          TYA
1800          PHA
1820 ;++++ INITIALISE VARIABLES ++++
1840          JSR SETUP
1860 ;++++PERFORM  MEM1=X(I)*CS-Y(I)*SN
1880 START
1890          LDA XILO
1900          LDY XIHI
1910          JSR MOVFM   ; FAC = X(I)
1920          LDA CSLO
1930          LDY CSHI
1940          JSR FMULT   ; FAC = X(I)*CS
1950          LDX #<MEM1
1960          LDY #>MEM1
1970          JSR MOVMF   ; MEM1=X(I)*CS
1980          LDA YILO
1990          LDY YIHI
2000          JSR MOVFM   ; FAC = Y(I)
2010          LDA SNLO
2020          LDY SNHI
2030          JSR FMULT   ; FAC = Y(I)*SN
```

```
2040          LDA #<MEM1
2050          LDY #>MEM1
2060          JSR FSUB    ; FAC = MEM1-FAC
2070          LDX #<MEM1
2080          LDY #>MEM1
2090          JSR MOVMF   ; MEM1= FAC
2110 ;++++PERFORM MEM2=Y(I)*CS+X(I)*SN
2130          LDA YILO
2140          LDY YIHI
2150          JSR MOVFM   ; FAC = Y(I)
2160          LDA CSLO
2170          LDY CSHI
2180          JSR FMULT   ; FAC = Y(I)*CS
2190          LDX #<MEM2
2200          LDY #>MEM2
2210          JSR MOVMF   ; MEM2= Y(I)*CS
2220          LDA XILO
2230          LDY XIHI
2240          JSR MOVFM   ; FAC = X(I)
2250          LDA SNLO
2260          LDY SNHI
2270          JSR FMULT   ; FAC = X(I)*SN
2280          LDA #<MEM2
2290          LDY #>MEM2
2300          JSR FADD    ; FAC = MEM2+FAC
2310          LDX #<MEM2
2320          LDY #>MEM2
2330          JSR MOVMF   ; MEM2= FAC
2350 ;++++PERFORM X(I)=MEM1:Y(I)=MEM2
2370          LDA #<MEM1
2380          LDY #>MEM1
2390          JSR MOVFM   ; FAC = MEM1
2400          LDX XILO
2410          LDY XIHI
2420          JSR MOVMF   ; X(I)= FAC
2430          LDA #<MEM2
2440          LDY #>MEM2
2450          JSR MOVFM   ; FAC = MEM2
2460          LDX YILO
2470          LDY YIHI
2480          JSR MOVMF   ; Y(I)=FAC
2500 ;++++TEST END LOOP
2520          DEC TEMPNP
2530          BEQ CONTIN
2550 ;++++INCREMENT ARRAY POINTERS
2570          LDA #$05
2580          CLC
2590          ADC XILO
2600          STA XILO
2610          BCC XNOHI
2620          INC XIHI
2630 XNOHI
2640          LDA #$05
2650          CLC
2660          ADC YILO
2670          STA YILO
2680          BCC YNOHI
2690          INC YIHI
2700 YNOHI
2710          JMP START
2730 ;++++ CLEAR/INIT SCREEN
2750 CONTIN
2760          LDA #$01
2770          STA $C100
2780          STA $C101
2790          STA $C102
2800          JSR $C10E ; INIT HIRES
2820 ;++++ PLOT LINES FROM E%(I,J)
2840 ; INITIALISE VARIABLES
2860          JSR SETUP
2870          LDA #$01
2880          STA IINDEX     ; I=1
2890          STA JINDEX     ; J=1
2900          LDA EBASLO     ; STORE E%
2910          STA ZPTEMP     ; POINTER
2920          LDA EBASHI     ; ON ZERO
2930          STA ZPTEMP+1   ; PAGE
2950 ; OFF WE GO - START GIANT I,J LOOP
2970 NEXTIJ
2980          LDY JINDEX
2990          LDA (ZPTEMP),Y ;GET E%(I,J)
3000          BNE DOIT
3010          JMP ONWARD
3030 ;++++PERFORM  X1%=X(I)+159
3040 ;    MLO=X1% LO:MHI=X1% HI
3060 DOIT
3070          LDA XILO
3080          LDY XIHI
3090          JSR MOVFM       ; FAC = X(I)
3100          LDA #<MEM1
3110          LDY #>MEM1
3120          JSR FADD        ; FAC = X(I)+159
3130          JSR FLPINT      ; .Y/.A = INT(FAC)
3140          STY MLO         ; X1% LO
3150          STA MHI         ; X1% HI
3170 ;++++PERFORM  X2%=X(J)+159
3180 ;    NLO=X2% LO:NHI=X2% HI
3200          LDA XJLO
3210          LDY XJHI
3220          JSR MOVFM       ; FAC = X(J)
3230          LDA #<MEM1
3240          LDY #>MEM1
3250          JSR FADD        ; FAC = X(J)+159
3260          JSR FLPINT      ; .Y/.A = INT(FAC)
3270          STY NLO         ; X2% LO
3280          STA NHI         ; X2% HI
3300 ;++++PERFORM Y1%=99-Z(I
3320          LDA ZILO
3330          LDY ZIHI
3340          JSR MOVFM       ; FAC =Z(I)
3350          LDA #<MEM2
3360          LDY #>MEM2
3370          JSR FSUB        ; FAC = 99-Z(I)
3380          JSR FLPINT      ; .Y/.A = INT(FAC)
3390          STY Y1          ; Y1%
3410 ;++++PERFORM Y2%=99-Z(J)
3430          LDA ZJLO
3440          LDY ZJHI
3450          JSR MOVFM       ; FAC =Z(J)
3460          LDA #<MEM2
3470          LDY #>MEM2
3480          JSR FSUB        ; FAC = 99-Z(J)
3490          JSR FLPINT      ; .Y/.A = INT(FAC)
3500          STY Y2          ; Y2%
3520 ; GET READY FOR LINSUB
3540 ;TWO BYTE COMPARISON FOR X1%=X2%
3560          LDA MLO
3570          CMP NLO
3580          BNE NOPE
3590          LDA MHI
3600          CMP NHI
3610          BNE NOPE
3630 ;NOW COMPARE Y1=Y2
3650          LDA Y1
3660          CMP Y2
3670          BNE NOPE
3690 ; ALL EQUAL SO AVOID LINESUB
3710          JMP ONWARD
3720 NOPE
3730          JSR LINSUB     ;PLOT LINE
3750 ; INCREMENT J POINTERS
3770 ONWARD
3780          LDA IINDEX     ;FIRST TEST
3790          CMP JINDEX     ;J=<I ?
3800          BEQ NEXTI
3820 ;INCREMENT XJLO/XJHI
3840          LDA #$05
3850          CLC
3860          ADC XJLO
3870          STA XJLO
3880          BCC XJNOHI
3890          INC XJHI
3900 XJNOHI
3920 ;INCREMENT ZJLO/ZJHI
3940          LDA #$05
3950          CLC
3960          ADC ZJLO
3970          STA ZJLO
3980          BCC ZJNOHI
3990          INC ZJHI
4000 ZJNOHI
4020 ;INCREMENT E% POINTER BY 1
4040          INC ZPTEMP
4050          BNE NOHIBY
4060          INC ZPTEMP+1
4070 NOHIBY
4080          INC JINDEX
4090          JMP NEXTIJ
4110 ; INCREMENT I POINTERS
4130 NEXTI
4140          LDA NP         ;FIRST TEST
4150          CMP IINDEX     ;I=<NP ?
4160          BEQ EXIT
4180 ; INCREMENT XILO/XIHI
4200          LDA #$05
4210          CLC
4220          ADC XILO
4230          STA XILO
4240          BCC XINOHI
4250          INC XIHI
4260 XINOHI
4280 ; INCREMENT ZILO/ZIHI
4300          LDA #$05
4310          CLC
4320          ADC ZILO
4330          STA ZILO
4340          BCC ZINOHI
4350          INC ZIHI
4360 ZINOHI
4380 ; INCREMENT E% POINTERS (TRICKY)
4400          LDA #$01       ; WHEN INC I
4410          CLC            ; MUST ADD
4420          ADC NP         ; 2*(NP+1)-(I-1)
4430          ASL A          ; TO THE ZP
4440          SEC            ; POINTER TO
4450          SBC IINDEX     ; E%
4460          CLC
4470          ADC #$01
4480          CLC
4490          ADC ZPTEMP
4500          STA ZPTEMP
4510          LDA ZPTEMP+1
```

```
4520          ADC #$00
4530          STA ZPTEMP+1
4550 ; REINITIALISE XJLO/XJHI
4570          LDA XBASLO
4580          STA XJLO
4590          LDA XBASHI
4600          STA XJHI
4620 ; REINITIALISE ZJLO/ZJHI
4640          LDA ZBASLO
4650          STA ZJLO
4660          LDA ZBASHI
4670          STA ZJHI
4690 ; REINITIALISE JINDEX
4710          LDA #$01
4720          STA JINDEX
4740 ; INCREMENT IINDEX
4760          INC IINDEX
4770          JMP NEXTIJ
4790 ; GIANT LOOP ENDS HERE
4800 ;
4810 ;++++ PULL REGISTERS ++++
4820 ;
4830 EXIT
4840          PLA
4850          TAY
4860          PLA
4870          TAX
4880          PLA
4890          RTS
4930 ;++++ SUBROUTINE
4940 ;
4950 SETUP
4960          LDA XBASLO
4970          STA XILO
4980          STA XJLO
4990          LDA XBASHI
5000          STA XIHI
5010          STA XJHI
5020          LDA YBASLO
5030          STA YILO
5040          LDA YBASHI
5050          STA YIHI
5060          LDA NP
5070          STA TEMPNP
5080          LDA ZBASLO
5090          STA ZILO
5100          STA ZJLO
5110          LDA ZBASHI
5120          STA ZIHI
5130          STA ZJHI
5140          LDY #159
5150          JSR SNGFT     ; FAC = 159
5160          LDX #<MEM1
5170          LDY #>MEM1
5180          JSR MOVMF     ; MEM1=159 IN FPT
5190          LDY #99
5200          JSR SNGFT     ; FAC = 99
5210          LDX #<MEM2
5220          LDY #>MEM2
5230          JSR MOVMF     ; MEM2=99 IN FTP
5240          RTS
```

BASIC Test Program

```
1000 REM**PLOT ROTATING CUBES**
1010 IFA=0THENA=1:LOAD"PLOTSUB.HEX",8,1
1020 IFA=1THENA=2:LOAD"LINESUB.HEX",8,1
1030 IFA=2THENA=3:LOAD"II-ROT.HEX",8,1
1040 PRINTCHR$(147)"WAIT 17 SECS"
1050 REM**DIMENSION ARRAYS**
1060 NP=24:REM NUMBER OF POINTS
1070 DIM X(NP),Y(NP),Z(NP)
1080 DIM E%(NP,NP):REM EDGE CONNECTIONS
1090 REM**INITIALISE ARRAYS**
1100 REM--INITIAL CUBE COORDINATE DATA
1110 DATA  50, 50, 50:REM----------/1
1120 DATA -50, 50, 50:REM TOP FOUR /2
1130 DATA -50,-50, 50:REM POINTS   /3
1140 DATA  50,-50, 50:REM----------/4
1150 DATA  50, 50,-50:REM----------/5
1160 DATA -50, 50,-50:REM BOT FOUR /6
1170 DATA -50,-50,-50:REM POINTS   /7
1180 DATA  50,-50,-50:REM----------/8
```

The following section of program reads in the data defined above into the arrays X(), Y() and Z(). Note that for the small cubes, the initial data values are reduced by a factor of 0.3

```
1190 REM**READ IN DATA FOR 3-CUBES**
1200 FORI=1TO8:REM CENTRE BIG CUBE
1210 READX(I),Y(I),Z(I)
1220 NEXT
1230 RESTORE
1240 FORI=9TO16:REM RIGHT SMALL CUBE
1250 READX,Y,Z
1260 X(I)=.3*X+120:Y(I)=.3*Y:Z(I)=.3*Z
1270 NEXT
1280 RESTORE
1290 FORI=17TO24:REM LEFT SMALL CUBE
1300 READX,Y,Z
1310 X(I)=.3*X-120:Y(I)=.3*Y:Z(I)=.3*Z
1320 NEXT
```

The next section rotates the co-ordinates of the points defining the three cubes to gain a 45° perspective

```
1330 REM**ROTATE SPACE ABOUT X-AXIS π/4
1340 FORI=1TONP
1350 Y(I)=Y(I)*COS(π/4)-Z(I)*SIN(π/4)
1360 Z(I)=Z(I)*COS(π/4)+Y(I)*SIN(π/4)
1370 NEXT
1380 REM**ROTATE SPACE ABOUT Z-AXIS π/4
1390 FORI=1TONP
1400 X(I)=X(I)*COS(π/4)-Y(I)*SIN(π/4)
1410 Y(I)=Y(I)*COS(π/4)+X(I)*SIN(π/4)
1420 NEXT
```

The section following defines the points to be connected within the figure for each of the three cubes using the adjacency array E%(,)

```
1430 REM**EDGE CONNECTION DATA**
1440 REM-MIDDLE BIG CUBE-
1450 E%(1,2)=1:REM 1 CONNECTED TO 2
1460 E%(2,3)=1:E%(3,4)=1:E%(4,1)=1
1470 E%(5,6)=1:REM BOT SQUARE
1480 E%(6,7)=1:E%(7,8)=1:E%(8,5)=1
1490 E%(5,1)=1:REM TOP TO BOT EDGES
1500 E%(6,2)=1:E%(7,3)=1:E%(8,4)=1
1510 REM-----------------
1520 REM-RIGHT SMALL CUBE
1530 FORI=9TO16:FORJ=9TO16
1540 E%(I,J)=E%(I-8,J-8)
1550 NEXT:NEXT
1560 REM-LEFT SMALL CUBE
1570 FORI=17TO24:FORJ=17TO24
1580 E%(I,J)=E%(I-8,J-8)
1590 NEXT:NEXT
1600 REM**SYMMETRISE E%(I,J)**
1610 FORI=1TONP:FORJ=1TONP
1620 IFE%(I,J)<>0THENE%(J,I)=1
1630 NEXT:NEXT
```

Now the program calls the machine code to perform the necessary rotation calculations and draw the new figure. This call is made repeatedly from within a loop rotating the three cubes through 360° before ending

```
1640 REM#########################
1650 REM##PLOT ROTATING CUBE##
1660 SA=2*π/45:CS=COS(SA):SN=SIN(SA)
1670 GOSUB1790:REM INITIALISE
1680 FOR A=0 TO 2*π STEP SA
1690 SYS50536:REM ROTATE
1700 NEXTA:REM NEXT ANGLE
1710 GETA$:IFA$=""THEN1710
1720 REM#########################
1730 GOSUB1750:REM RESET SCREEN
1740 END
1750 REM**RESET SCREEN**
1760 POKE49408,0:SYS49422
1770 PRINTCHR$(147)
1780 RETURN
1790 REM**INITIALISE ROTSUB**
1800 REM**************************
1810 REM**N.B. DO NOT DEFINE NEW**
1820 REM**VARIABLES BETWEEN THIS**
1830 REM** SUBROUTINE & CALLING **
1840 REM**  MACHINE CODE ROTATE.**
1850 REM** AS THIS WOULD CHANGE **
1860 REM** BASE ADDRESS ARRAYS  **
1870 REM**************************
1880 X(1)=X(1):REM X(1) CURRENT VAR
1890 POKE50500,PEEK(71):REM X(1)LO
1900 POKE50501,PEEK(72):REM X(1)HI
1910 Y(1)=Y(1):REM Y(1) CURRENT VAR
1920 POKE50502,PEEK(71):REM Y(1)LO
1930 POKE50503,PEEK(72):REM Y(1)HI
1940 Z(1)=Z(1):REM Z(1) CURRENT VAR
1950 POKE50504,PEEK(71)
1960 POKE50505,PEEK(72)
1970 POKE50506,NP:REM NUMBER OF POINTS
1980 CS=CS:REM MAKE CS CURRENT VAR
1990 POKE50507,PEEK(71)
2000 POKE50508,PEEK(72)
2010 SN=SN:REM MAKE SN CURRENT VAR
2020 POKE50509,PEEK(71)
2030 POKE50510,PEEK(72)
2040 E%(1,1)=E%(1,1):REM E% CURRENT VAR
2050 POKE50511,PEEK(71)
2060 POKE50512,PEEK(72)
2070 RETURN
```

PASSWORD

Alphanumeric combinations of characters enabling users to log on to a computer system are called *passwords,* and are widely used in commercial applications. They prevent unauthorised persons from gaining access to the system who might 'steal' valuable computer time or examine confidential information held on the computer.

When a user attempts to log on to a system requiring a password, the computer will first display a prompt asking for it. When the user has typed in the password, the computer will then compare the word with that which is held in its memory. If the two words are identical, the user will be allowed access to the rest of the system. Even if someone manages to log on to the computer, there may be certain other areas of the computer that require further passwords in order to access them.

Many large computer database and network systems allow you to define your own passwords to ensure personal security. Initially, you will be given a password by the database and, once logged on, will be able to change it to your own specification. However, this system does have its drawbacks. Many large companies have discovered that their data processing departments pin the passwords on the wall.

Other problems result from the lack of imagination used in attempts to customise passwords. Commonly used passwords include the user's name, birthday, the name of the area of the database one intends to enter, 'Fred' or, simply, 'Password'. This sort of obvious password is a gift to hackers (see page 716) as it makes the problem of cracking the codes that much easier.

PATCH

When a program does not run correctly it is often necessary to add a *patch,* which is a piece of code enabling the program to execute properly. Although they are sometimes necessary to help correct minor mistakes in a completed listing, patches tend to be frowned upon by professional programmers. This is because patches tend to be used as a quick and simple method of getting the program to work. As a result, frequent users tend to ignore structure and security in favour of eliminating bugs. A large amount of patching (which should not be needed if the program is written properly) soon becomes self-defeating, as the patches will become confused and the resulting program will be horrendously difficult to debug.

Hardware patches are pieces of cable that join various components within an electronic circuit. This kind of patch, like its software counterpart, is sometimes used to make a circuit operate correctly. More usefully, however, they are used to make temporary connections onto a patchboard rather like an old-fashioned telephone switchboard. This type of patching is used in synthesisers, certain types of computers and in connecting devices together in a network.

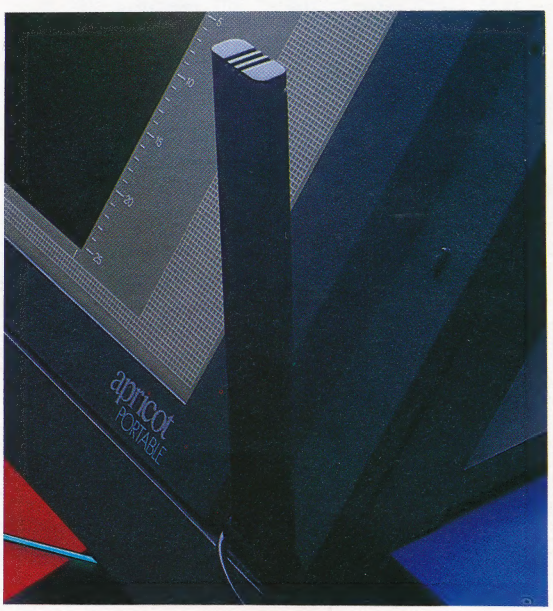
PATTERN RECOGNITION

In the field of artificial intelligence, *pattern recognition* is an area of intense research. Although the term is usually taken to refer to vision and speech recognition, the concept is more fundamental. The computer does not have 'eyes' or 'ears', but it can be connected to cameras and microphones that convert light and sound into electrical signals and transmit them to the computer. Thus, pattern recognition is based on the interpretation and comparing of these signals to patterns held in the computer's memory.

If the signal pattern and memory pattern match, the computer will 'recognise' the pattern. This is known as the 'template' method of pattern recognition — the computer will be programmed with, for example, the shape of a square in its memory and make a direct comparison with this. Although this is the simplest method, it is rather crude and uses vast quantities of memory space.

A more advanced area of pattern recognition is the 'analytical' method. In this system, the computer will examine the shape transmitted from a camera, deduce that the corners have right angles, that the edges are of equal length and conclude that the pattern is a square. This system, although requiring much more complex programming techniques, is more adaptable and requires less memory space.

Once a computer has been programmed to recognise certain patterns, the next step is to enable it to identify variations of a basic pattern. The field where this is most critical is *speech recognition.* For example, although a computer may have been programmed to recognise the spoken word 'run', individual users will pronounce the word in different ways. Thus, the template of 'run' cannot be too restrictive for a reasonable rate of successful recognitions. The answer is to adjust the template to take in a band of different pattern frequencies, all of which mean 'run'. The problem here, of course, is the enormous amount of memory required to store just a few words!



Speak Up!
The Apricot Portable is one of the first commercial computers to incorporate a speech recognition facility as standard. Simple words, such as a series of numbers, are input via the microphone provided. Although this is a long way from the sort of interactive conversation with computers portrayed in films like 2001, A Space Odyssey, it is an important first step in the development of machines that understand speech

HAVE YOU ORDERED YOUR VOLUME SIX BINDER YET?

FOLD 2

IF YOU HAVE NOT ASKED FOR THE BINDERS TO BE SENT TO YOU AUTOMATICALLY, DO SO NOW, AND ENSURE THAT YOUR COPIES OF THE HOME COMPUTER ADVANCED COURSE ARE KEPT PROPERLY BOUND AND IN GOOD CONDITION FOR YEARS TO COME.

BY TICKING THE BOX OPPOSITE, YOU WILL BE SENT BINDER NUMBER 6.

☐ PLEASE SEND ME MY VOLUME 6 BINDER NOW. I ENCLOSE A CHEQUE/POSTAL ORDER FOR £3.95 (WHICH INCLUDES POSTAGE AND PACKING).

FOLD 3

BY TICKING THE OTHER BOX *AS WELL*, YOU WILL BE SENT SUBSEQUENT BINDERS FOR YOUR COLLECTION.

☐ I WOULD ALSO LIKE TO RECEIVE FUTURE BINDERS AS THEY ARE ISSUED. I UNDERSTAND THAT I WILL RECEIVE A PAYMENT ADVICE FOR £3.95 (WHICH INCLUDES POSTAGE AND PACKING) WITH EACH BINDER. IF I AM NOT SATISFIED WITH THE BINDER, I CAN RETURN IT TO YOU, WITHIN 14 DAYS, AND OWE NOTHING.

NO STAMP NECESSARY.

JUST FOLD UP THE PAGE AS INDICATED, REMEMBERING TO ENCLOSE YOUR CHEQUE/POSTAL ORDER MADE PAYABLE TO ORBIS PUBLISHING, AND SEND TO US TODAY.

FOLD 4

FOLD 4

I enclose a cheque/postal order made payable to: Orbis Publishing Ltd. for a total of £____ which I understand includes the cost of postage and packing.

NB: Please allow 28 days for the delivery of your binders.

When you have completed the order form fill in your name and address in the space provided.

Then cut along the dotted line to detach the page, enclose your cheque/postal order, and fold the page carefully – following the instructions to complete the reply paid envelope.

NO STAMP NECESSARY.

FOLD 1

IF YOU ALREADY HAVE AN ACCOUNT NO. FOR YOUR BINDERS PLEASE FILL IT IN HERE.

Complete the section below with one letter or figure per space.

		INITIALS	SURNAME
MR			
MRS			
MISS			

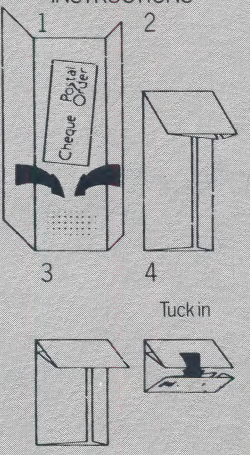
ADDRESS & POST CODE

TELEPHONE NO. INCLUDING STD CODE OR EXCHANGE NAME

FOLD 1

FOLD 4

FOLDING INSTRUCTIONS



ARE YOU LOOKING AFTER YOUR COPIES.

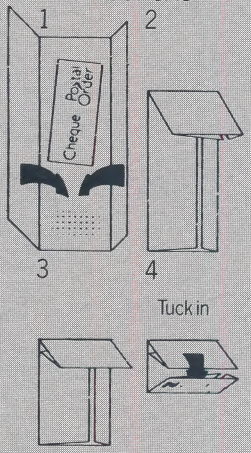
FOLD 2

FOLD 3

FOLD 4

FOLD 4

FOLDING INSTRUCTIONS



FOLD 1

Postage will be paid by licensee

Do not affix Postage Stamps if posted in Gt. Britain, Channel Islands or N. Ireland.

BUSINESS REPLY SERVICE
Licence No. SW4035.

The Home Computer
Advanced Course Binders
Orbis House, 20-22 Bedfordbury
London WC2N 4BR

2

FOLD 1

GUARANTEE GUARANTEE GUARANTEE GUARANTEE GUARANTEE

GUARANTEE

If you are not entirely satisfied with your binder, send it back immediately and it will be either exchanged, or, if you prefer, your money will be refunded in full.

GUARANTEE